CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Facultad de Informática

TRABAJO FIN DE GRADO

Sistema de soporte al desarrollo de un planificador de misiones en un vehículo aéreo no tripulado

Autor: Adrián Martínez Alcaraz

Director: Martín Molina González

MADRID, JUNIO DE 2014

ÍNDICE

RESUMEN

En esta memoria se describe el trabajo de construcción de una arquitectura software diseñada para facilitar el desarrollo un planificador de misión de un vehículo aéreo no tripulado (UAV), con el fin de que éste alcance los objetivos marcados en la competición internacional de robótica IARC (séptima edición).

A lo largo de la memoria, se describe en primer lugar, una revisión de técnicas de robótica inteligente aplicadas a la construcción de vehículos aéreos no tripulados, en el que se ven los diferentes paradigmas de programación de la robótica inteligente y la clasificación de dichos robots aéreos, dependiendo de su autonomía. Este descripción finaliza con la presentación del problema correspondiente a la competición IARC.

A continuación se describe el diseño realizado para soporte al desarrollo de un planificador de misiones de UAVs, con simulación de comportamiento de vehículos robóticos y visualización 3D con movimiento. Finalmente, se muestran las pruebas que se han realizado para validar la construcción de dicha arquitectura software.

ABSTRACT

In this report it is presented the construction of a software architecture, designed to facilitate the development of a mission planner for an unmanned aerial vehicle (UAV), so that it reaches the goals set in the International Aerial Robotics Competition - IARC (seventh edition).

Throughout this report, it is described first, a review of intelligent robotics techniques applied to the construction of unmanned aerial vehicles, where different paradigms of intelligent robotics are seen, along with a classification of such aerial robots, depending on their autonomy. Description ends with the presentation of the problem corresponding to the IARC competition.

Following, it is described the design made to satisfy the support to the development of a mission planner for UAV´s, with a simulation of the robotics vehicles' behaviours and a 3D display with motion. Finally, we will deal with the tests that have been conducted to validate the construction of the software architecture.

1. INTRODUCCIÓN

El presente Trabajo de Fin de Grado tiene como objetivo general la construcción de una arquitectura software para dar soporte al desarrollo de un planificador de misiones de un vehículo no tripulado (UAV: *Unmmaned Aerial Vehicle*), diseñado para participar en la competición internacional de robótica IARC (séptima edición).

Puesto que es difícil realizar pruebas directamente sobre el UAV, debido a la imposibilidad de tener físicamente todos los elementos que deben interactuar con éste de forma anticipada en el proceso de construcción del mismo, surge la necesidad de crear un entorno simulado.

Este entorno de simulación es útil para experimentar con diferentes prototipos de planificadores de misión, con el fin de realizar una elección adecuada que cumpla con los objetivos marcados en la competición, antes de ser integrado en el hardware real.

1.1 Objetivos

Los objetivos planteados en el presente trabajo se han orientado a cubrir las diferentes fases de construcción de la arquitectura del entorno de simulación para ayuda a la construcción del planificador de misiones del UAV. La consecución de este objetivo se desglosa de la siguiente manera:

- **Análisis:** Estudio detallado del problema sobre el que se realiza el trabajo. Ello engloba el estudio de las diferentes especificaciones y problemáticas que se plantean para llegar a la construcción de la arquitectura software. Este estudio contempla los diferentes paradigmas de la robótica inteligente, las reglas del concurso IARC junto con métodos y herramientas para implementar el objetivo final.

- **Diseño:** En esta fase se trata de realizar el diseño de cada uno de los componentes software que forman la arquitectura. De especial importancia es en esta fase el módulo de visualización, pues éste depende de las diversas herramientas con las que se pueda implementarla, además es el modulo que permitirá visualizar la actitud del resto de componentes.

- **Implementación:** Se trata de elegir un lenguaje de programación que permita una programación modular de esta arquitectura y que además se adapte a las

diferentes herramientas software de robótica que ayuden a la visualización de la simulación de dicha arquitectura.

- **Validación:** Esta tarea tiene como fin realizar pruebas necesarias para verificar que el conjunto de los elementos que conforman la arquitectura cumplen con los objetivos del trabajo.

1.2 Organización de la memoria

Esta memoria se divide en 4 capítulos principales. En el primero de ellos se realiza la descripción del problema, para esta descripción se pasan por puntos como la robótica inteligente, los diferentes niveles de autonomía que se podrán adoptar, los paradigmas de programación existentes dentro de la robótica inteligente, grupos de desarrollo actualmente trabajando en el campo de los UAV y por último, de manera más detallada las reglas de la competición IARC. En el siguiente capítulo veremos la estrategia de diseño llevada a cabo para construir la arquitectura software y como se ha llevado a cabo la implementación de este diseño, mediante un sistema software de robótica. Éste permitirá la comunicación de los diferentes módulos que conforman la arquitectura y la visualización de la interacción entre éstos. Por último, el cuarto capítulo contiene las distintas pruebas realizadas al conjunto de la arquitectura para llegar a su validación. Finalmente en el capítulo dedicado a las conclusiones se resumen los resultados obtenidos.

2. DESCRIPCIÓN DEL PROBLEMA

En este capítulo se presenta una visión general de la robótica inteligente, cómo se clasifican los robots en cuanto a autonomía, y de cómo ha evolucionado debido a la investigación de los diferentes grupos de desarrollo. Se incide especialmente en las reglas de la competición internacional de robótica IARC, mostrando también su evolución histórica.

2.1 Introducción

Según la Real Academia de la Lengua Española el significado de robótica es una *"Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales."*

Se debe realizar una diferenciación dentro de los posibles tipos de robótica antes de adentrar de manera más en profunda en un tipo en particular. Existen diferentes tipos de robots dependiendo de la aplicación de cada uno de éstos o según el grado de sustitución de las personas que pueden alcanzar, como contempla la RAE en su definición. Estos son:

- **Robótica Industrial:** Es la parte de la Ingeniería que se dedica a la construcción de máquinas capaces de realizar tareas mecánicas y repetitivas de una manera muy eficiente y con costes reducidos.

- **Robótica de Servicio:** Es la parte de la Ingeniería que se centra en el diseño y construcción de máquinas capaces de proporcionar servicios directamente a los miembros que forman una sociedad.

- **Robótica Inteligente:** Se encarga del desarrollo de robots manipuladores o sistemas mecánicos multifuncionales controlados por computadores capaces de relacionarse con su entorno o a través de sensores y tomar decisiones en tiempo real, el cual es el concepto de la inteligencia artificial.

- **Robótica Humanoide:** Es la parte de la ingeniería que se dedica al desarrollo de sistemas robotizados que tratan de imitar determinadas peculiaridades del ser humano.

El tipo de robótica en el que se basa el presente trabajo y que se ve a lo largo de esta memoria, es la robótica inteligente.

2.2 Robótica Inteligente

La robótica inteligente ha tenido una gran evolución a lo largo de los últimos años. Principalmente debido a los sistemas UAV que han visto incrementado el número de aplicaciones en los que pueden ser usados, tanto militares como civiles. Algunas de las aplicaciones en las que destacan estos sistemas son la vigilancia, reconocimiento e inspección aérea, entre otras muchas funciones en las que la actuación del ser humano resulta compleja y peligrosa.

Gracias esta reducción del riesgo y una mayor confianza en el éxito del cumplimento de las misiones por parte de estos sistemas, están haciendo que su desarrollo e inversión se vea incrementado. Según diversos estudios de mercado realizados (Visiongain, 2009), (Dickerson, 2007) se ha predicho que el mercado emergerá y se expandirá considerablemente a lo largo de la siguiente década, algo que ya se esta viendo en la actualidad, con los UAV existentes para uso comercial.

Dentro de la robótica inteligente se debe hacer una diferenciación de los diversos sistemas que se pueden encontrar, éstos son los sistemas automáticos, sistemas autónomos y sistemas inteligentes.

- Un robot automático realiza una tarea que se le ha programado y no tiene ninguna capacidad de razonamiento y decisión.

- El robot autónomo tiene la capacidad de elegir las tareas a realizar para alcanzar el objetivo o los objetivos que se le han asignado, dependiendo de la percepción que este tenga del mundo exterior, sin que ningún agente externo intervenga en su control.

- El robot inteligente es un sistema autónomo al que además se le otorga la habilidad de decidir o generar sus metas u objetivos dependiendo de sus propias motivaciones.

La arquitectura de control que se ve a lo largo de esta memoria trata de dotar autonomía a un UAV, con el fin de que este consiga la misión que se le asigne, convirtiéndose así en un robot autónomo.

2.3 Niveles de Autonomía

El nivel de autonomía describe el grado que un robot toma y ejecuta diferentes decisiones. Tom Sheridan propone es su trabajo (Sheridan, 1992) 10 niveles de grado de autonomía, que van desde el robot controlado completamente por un ser humano hasta el robot completamente autónomo que no requiere aprobación de sus acciones:

1. El ordenador no ofrece asistencia, el humano hace todo.
2. El ordenador ofrece una serie de acciones alternativas.
3. El ordenador estrecha la selección a unas pocas elecciones.
4. El ordenador sugiere una única acción
5. El ordenador ejecuta la acción si el humano la aprueba.
6. El ordenador permite al humano un tiempo de aceptación de la acción antes de ser ejecutada automáticamente.
7. El ordenador ejecuta automáticamente la acción informando al humano.
8. El ordenador informa al humano después de ejecutar la acción automáticamente, solo si el humano pregunta.
9. El ordenador informa al humano después de ejecutar la acción automáticamente, solo si el ordenador decide que debe hacerlo.
10. El ordenador decide todo y actúa automáticamente ignorando al humano.

A partir de la clasificación realizada por Sheridan surgieron otras revisiones, como la sugerida por Parasuraman (Parasuraman et al., 2000), el cuál se dio cuenta que dichas escalas podrían no ser aplicables al dominio total del problema, pero si a las diferentes tareas que se encuentran en dicho dominio. Por ello, sugirió un modelo de niveles basados en cuatro clases de funciones, obtención de información , análisis, decisión y selección de la acción a realizar y ejecución de la acción.

Posteriormente en 2002, el Laboratorio de investigación de la fuerza aérea de los Estados Unidos (US Air Force Research Laboratory – AFRL) presentaron los resultado de un estudio de investigación denominado “Autnomous Control Level - ACL” para medir el nivel de autonomía de un UAV mediante una tabla que contempla un total de 11 niveles de autonomía. Dicha tabla se basa en el concepto OODA, el cual contempla la percepción y conciencia de la situación (Observe), análisis y coordinación (Orient), decisión (Decide) y capacidad de actuación (Act).
En 2007 un grupo de trabajo financiado por el instituto nacional de estándares y tecnología estadounidense (NIST), desarrolló un marco de trabajo para definir los niveles de autonomía para sistemas no tripulados denominado ALFUS (Autonomy Levels For Unmanned Systems). Los niveles que conforman la tabla de este marco

están basados en una serie de métricas, con transiciones ligeras entre los diferentes. Se utilizan 3 unidades de medida para clasificarlos, independencia humana (Human Independence – HI), complejidad de la misión(Mission Complexity – MC) y complejidad del entorno (Environmental Complexity - EC). Estas métricas clasifican diversos factores como se muestra en la Figura 2.1.

ACL es aplicable principalmente a UAV´s de grandes dimensiones, que vuelan a altitudes muy altas y libres de obstáculos. Por otro lado, ALFUS contempla todos los sistemas de vehículos aéreos, de manera muy genérica, es decir, para todo tipo de UAV´s. Debido a esto, Kendoul (Kendoul, 2012) basándose en los anteriores marcos, plantea una clasificación especifica para sistemas aéreos no tripulados basados en hélices denominados RUAS (Rotorcraft Unmanned Aerial System), los cuales son sistemas que trabajan a baja altura y en diferentes entornos, surgiendo así el marco ALFURS (Autonomy Levels For Unmanned Rotorcraft Systems). Éste no solo es aplicable a un único tipo de UAV sino que aunque puede ser usado de manera genérica en otros tipos de UAV.

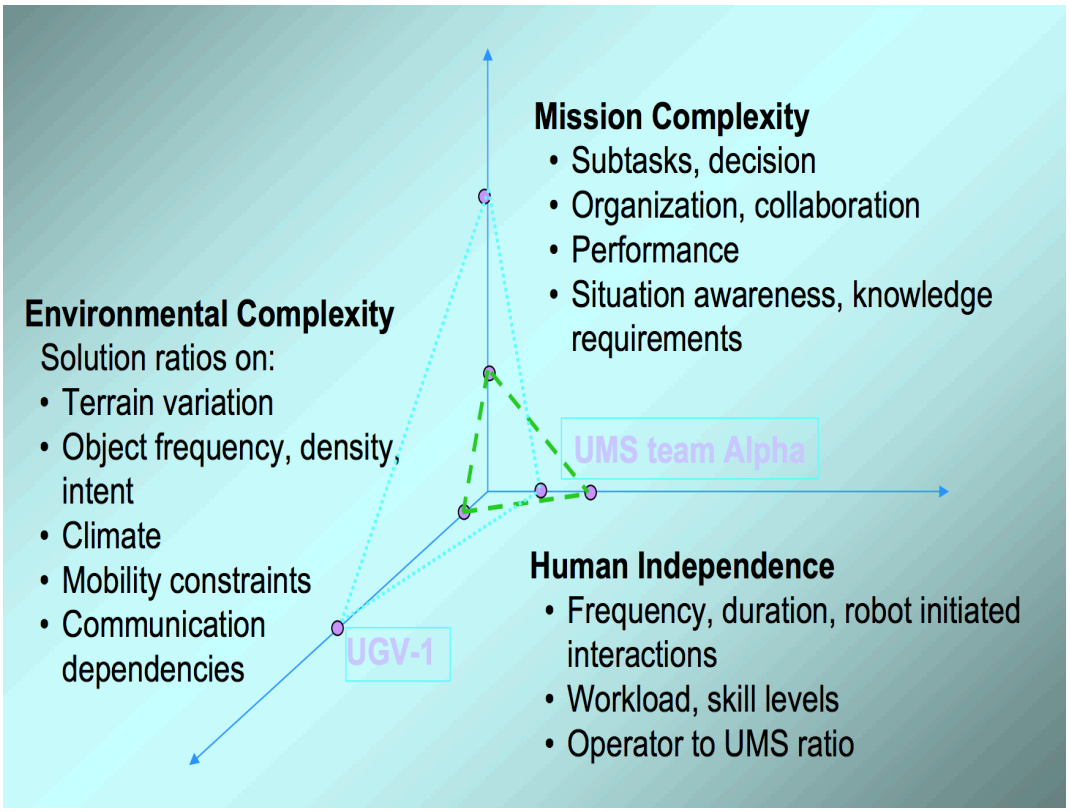

Figura 2.1. Métricas de clasificación para ALFUS

2.3.1 ALFURS - Autonomy Levels For Unmanned Rotorcraft Systems

Este marco se basa en las funciones que tiene el robot y le permiten en cierta manera ser autónomo, ALFURS denomina a estas funciones "AEF" (Autonomy Enably Functions). Dichas funciones las separa en tres categorías, Guiado, Navegación y Control de vuelo, resultando en el modelo GNC (Guidance, Navigation and Control), representado en la Figura 2.2. A continuación se muestra un poco más en detalle cuales son los aspectos de los que se encarga cada uno de los distintos sistemas que conforman el GNC.

- **Sistema de control de vuelo(Flight Control System):** Se encarga de actuar sobre los diferentes sistemas motrices del robot con el fin de obtener la salida esperada. Se trata de obtener unas leyes de control que permitan tras recibir una orden, acción a ejecutar u otra serie de inputs, producir unas señales que controlen la posición, velocidad, altitud u otros factores físicos que permitan al robot conseguir con éxito la acción deseada.

  Sistema de Navegación (Navigation System): Lleva a cabo el proceso de monitorización y control de movimiento del robot de un lugar a otro. Para ello debe producirse un proceso de obtención de datos y análisis, tanto de los diversos parámetros del estado del robot como del entorno que le rodea, con el fin de realizar de manera exitosa y segura, la misión de éste. La obtención de estos datos tiene como objetivo hacer que el robot consiga una conciencia de la situación o en ingles "Situational Awareness" según la definición de Endsley (Endsley, 1999), *" Es la percepción de los elementos existentes en el entorno dentro de un volumen de espacio y tiempo, la comprensión de su significado y la proyección de su estado en un futuro próximo"*. Por ello es necesario un entendimiento de los datos y una proyección de estos en un tiempo futuro, con el fin de predecir como afectará el entorno y el estado del robot a su propia operación. Los datos necesarios para llegar a esta conciencia de la situación se obtienen de diversos subsistemas:

  - *Sistema de sensores:* Este sistema agrupa los diversos dispositivos físicos que lleva a bordo el robot y se encargan de obtener información del propio robot y del entorno que le rodea. Éstos pueden ser muy diversos, sensores giroscópicos, acelerómetros, sensores de presión, cámaras, sistemas de flujo óptico, todos ellos proporcionan información en bruto para su análisis en los siguientes subsistemas.

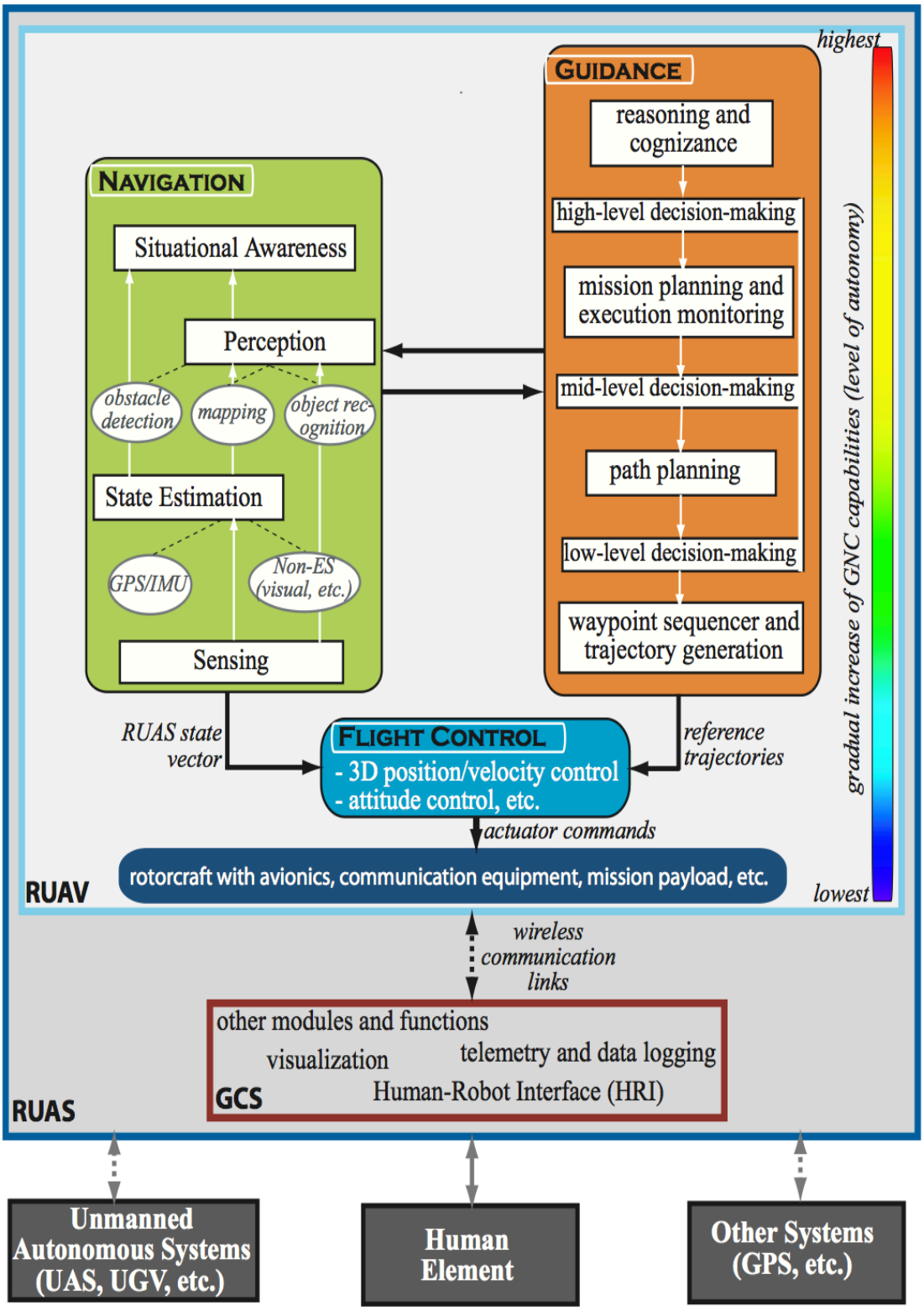

Figura 2.2. Modelo de arquitectura de un UAV basado en el modelo, propuesto en el marco ALFURS.

- *Estimador del estado:* Tras recibir la información del conjunto de sensores, realiza el proceso de estimación de las diferentes variables que afectan al estado del robot. Entre estas se encuentran la velocidad, aceleración, posición, dirección, altitud, etc.

- *Percepción:* Crea un modelo del entorno que le rodea por medio de la información que proviene del sistema de sensores. Esta percepción del entorno puede ser a su vez dividida en diferentes partes, cartografía, obstáculos, reconocimiento de objetos o detección de objetivos.

- **Sistema de Guiado (Guidance System):** Este es el sistema que lleva a cabo las tareas de planificación y toma de decisiones necesarias para alcanzar la misión o meta asignada al robot. Conforma el sistema cognitivo del robot, el cual suplanta al operador humano que lo controlaría. Recibe información del sistema de navegación y realiza la toma de decisión, la cual es descompuesta en ordenes de bajo nivel, como ordenes de ascenso y descenso, variación de velocidad, variación de trayectoria, etc. y son enviadas al sistema de control de vuelo. Para conseguir la planificación y toma de decisiones, el sistema de guiado lleva a cabo diversas subfunciones:

  - *Planificador de trayectoria:* Realiza la secuenciación de los diferentes puntos que tiene que seguir el UAV, basándose en la información de navegación acumulada que permiten encontrar el mejor y mas seguro camino para alcanzar una posición, una configuración o una tarea específica.

  - *Generación de trayectoria:* Tiene el papel de generar diversas funciones motoras como la posición o dirección dependiendo de si son físicamente posibles y cumplen con las especificaciones y restricciones del UAV. La información de salida de este submódulo puede ser usada directamente como referencia de trayectoria del controlador de vuelo.

  - *Planificador de la misión:* Se encarga de generar las metas, una ruta, coordinación y tiempo del UAV. Posteriormente se encarga de la selección de acciones dentro de un conjunto de diferentes escenarios, tras el análisis de la información recibida. Las decisiones pueden variar en un gran rango, desde un bajo nivel hasta decisiones de alto nivel.

Una vez visto en mas detalle las funciones GNC, es posible la inmersión en como se estructura la tabla de niveles de autonomía del marco ALFURS. Los niveles vienen determinados dependiendo de la complejidad de las funciones GNC que el UAV cumple, cuanto mayor es la complejidad de éstas, mayor es el nivel de autonomía.
Es posible establecer una relación entre las métricas del marco ALFUS (Mission Complexity – MC, Human Independence – HI, Environmental Complexity – EC) y las métricas del marco ALFURS. Una de las motivaciones de usar las funciones GNC para caracterizar el nivel de autonomía, es por la familiarización que tiene la comunidad de investigación de UAV´s con estos términos, además de describir los niveles de la manera más simple y entendible.

En la figura 2.3 se pueden observar los diferentes niveles propuestos en el marco ALFURS, así como su comparación con el marco ALFUS.

Para alcanzar uno de los niveles de autonomía vistos, la programación de los sistemas de los robots deben basarse en uno de los diferentes paradigmas de la robótica inteligente existentes, los cuales indican como se organiza la inteligencia de éstos, con el fin de conseguir la autonomía deseada.

2.4 Paradigmas de la robótica inteligente

Un paradigma es un conjunto de técnicas o teorías que caracterizan una aproximación a la solución de una clase de problemas. Es una manera de mirar al mundo, como tambíen una serie de herramientas para solucionar problemas. Ninguno de los paradigmas es el correcto, únicamente habrá problemas que se ajustan mejor a alguno de los diferentes paradigmas. Aplicando el paradigma correcto hará que la solución al problemas sea de una manera más fácil.

Conocer los paradigmas de la robótica es la clave para programar de manera exitosa un robot para una determinada aplicación. Actualmente existen tres paradigmas diferentes para organizar la inteligencia de los robots (Murphy 2000): paradigma jerárquico, paradigma reactivo y paradigma híbrido deliberativo/reactivo.

2.4.1 Paradigma jerárquico

Es el primero y más antiguo paradigma existente, fue el mas utilizado entre los años de 1967 a 1990. Este se basa en como las personas piensan de una manera introspectiva, como por ejemplo, una persona no planea como salir de una habitación o como trazar la

LEVEL	LEVEL DESCRIPTOR	GUIDANCE	NAVIGATION	CONTROL	ESI	EC	MC
10	Fully Autonomous	Human-level decision-making, accomplishment of most missions without any intervention from ES (100% ESI), cognizant of all within the operation range.	Human-like navigation capabilities for most missions, fast SA that outperforms human SA in extremely complex environments and situations.	Same or better control performance as for a piloted aircraft in the same situation and conditions.	*approaching 100% ESI*	*extreme environment*	*highest complexity, all missions*
9	Swarm Cognizance and Group Decision Making	Distributed strategic group planning, selection of strategic goals, mission execution with no supervisory assistance, negotiating with team members and ES.	Long track awareness of very complex environments and situations, inference and anticipation of other agents intents and strategies, high-level team SA.	Ability to choose the appropriate control architecture based on the understanding of the current situation/context and future consequences.	*high level ESI*	*difficult environment*	*collaborative, high complexity missions*
8	Situational Awareness and Cognizance	Reasoning and higher level strategic decision-making, strategic mission planning, most of supervision by RUAS, choose strategic goals, cognizance.	Conscious knowledge of complex environments and situations, inference of self/others intent, anticipation of near-future events and consequences (high fidelity SA).	Ability to change or switch between different control strategies based on the understanding of the current situation/context and future consequences.			
7	RT Collaborative Mission Planning	Collaborative mission planning and execution, evaluation and optimization of multi-vehicle mission performance, allocation of tactical tasks to each agent.	Combination of capabilities in levels 5 and 6 in highly complex, adversarial and uncertain environment, collaborative mid fidelity SA.	same as in previous levels (no-additional control capabilities are required)	*mid level ESI*	*moderate environment*	*mid complexity, multi-functional missions*
6	Dynamic Mission Planning	Reasoning, high-level decision making, mission driven decisions, high adaptation to mission changes, tactical task allocation, execution monitoring.	Higher-level of perception to recognize and classify detected objects/events and to infere some of their attributes, mid fidelity SA.	same as in previous levels (no-additional control capabilities are required)			
5	RT Cooperative Navigation and Path Planning	Collision avoidance, cooperative path planning and execution to meet common goals, swarm or group optimization.	Relative navigation between RUAS, cooperative perception, data sharing, collision detection, shared low fidelity SA.	Distributed or centralised flight control architectures, coordinated maneuvers.			
4	RT Obstacle/Event Detection and Path Planning	Hazard avoidance, RT path planning and re-planning, event driven decisions, robust response to mission changes.	Perception capabilities for obstacle, risks, target and environment changes detection, RT mapping (*optional*), low fidelity SA.	Accurate and robust 3D trajectory tracking capability is desired.			
3	Fault/Event Adaptive RUAS	Health diagnosis, limited adaptation, onboard conservative and low-level decisions, execution of pre-programmed tasks.	Most health and status sensing by the RUAS, detection of hardware and software faults.	Robust flight controller, reconfigurable or adaptive control to compensate for most failures, mission and environment changes.	*low level ESI*	*simple environment*	*low level tasks*
2	ESI Navigation (*e.g., Non-GPS*)	Same as in Level 1	All sensing and state estimation by the RUAS (no ES such as GPS), all perception and situation awareness by the human operator.	Same as in Level 1			
1	Automatic Flight Control	Pre-programmed or uploaded flight plans (waypoints, reference trajectories, etc.), all analyzing, planning and decision-making by ES.	Most sensing and state estimation by the RUAS, all perception and situational awareness by the human operator.	Control commands are computed by the flight control system (automatic control of the RUAS 3D pose).			
0	Remote Control	All guidance functions are performed by external systems (mainly human pilot or operator)	Sensing may be performed by the RUAS, all data is processed and analyzed by an external system (mainly human).	Control commands are given by a remote ES (mainly human pilot).	*0% ESI*	*lowest EC*	*lowest MC*

Figura 2.3. Tabla de niveles de autonomía. Acrónimos: ESI (External System Independence), EC (Environment Complexity), MC (Mission Complexity), ES (External System), SA (Situational Awareness), RT (Real-Time).

ruta dentro de ésta para alcanzar la puerta, sino que dispone de un esquema mental interno o comportamiento para saber como resolver dicha situación.
Dentro del paradigma jerárquico el robot percibe el mundo, planea la siguiente acción y entonces actúa como se representa en la Figura 2.4. En cada etapa el robot planea el

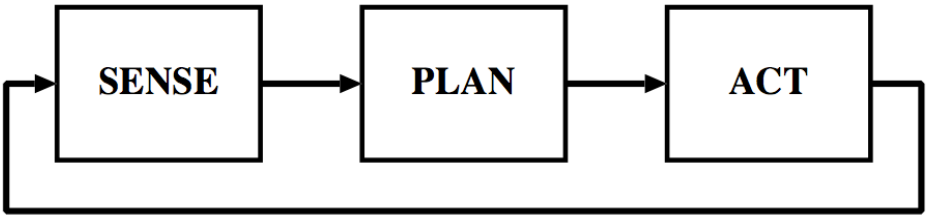

Figura 2.4. Organización del paradigma jerárquico.

El siguiente movimiento. Otra de las características de este paradigma es que toda la información percibida es recogida para crear un modelo del mundo único que es usado por el planificador. Este modelo del mundo genérico y cerrado puede dar problemas a la hora de actuar, pues este contiene todo lo que el robot necesita saber y si esta condición no se cumple el robot no podrá realizar la tarea asignada.

Como ejemplo de actuación de un robot dentro del paradigma jerárquico, éste primero abre los ojos, percibe el mundo y crea un mundo mental, posteriormente planifica las directivas a seguir para conseguir la meta y por último realiza dichas directivas. En la Figura 2.5 se muestran estas transiciones entre las diferentes primitivas de percibir, planificar y actuar.

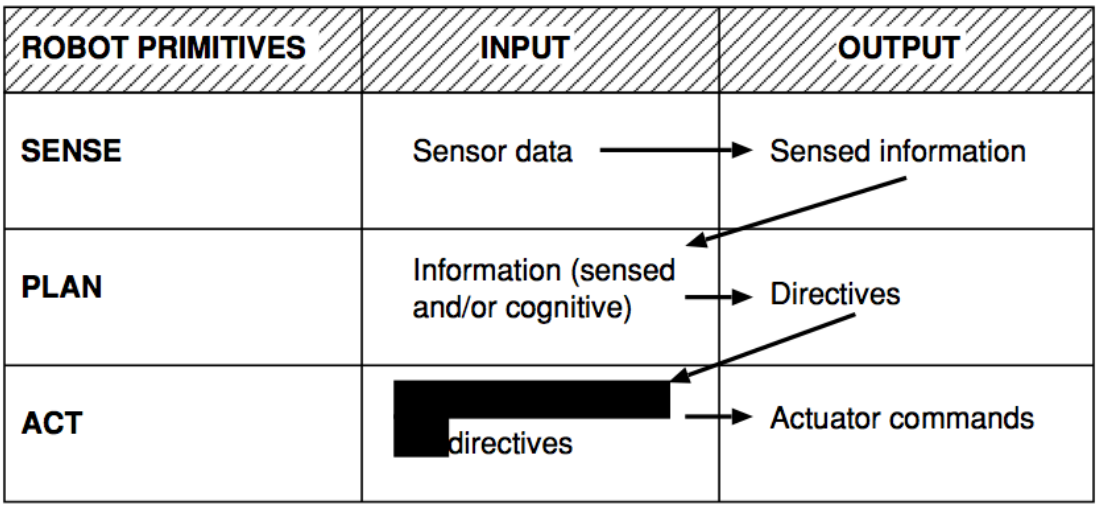

Figura 2.5. Transiciones de las primitivas del robot en el paradigma jerárquico

La principal desventaja de este paradigma es la planificación, pues en cada ciclo de actualización el robot debe actualizar el modelo del mundo y planificar. Estos algoritmos son muy lentos, además como la percepción y la actuación están separados, se elimina cualquier acción reactiva ante un estimulo imprevisto.

2.4.2 Paradigma reactivo

El paradigma reactivo fue la consecuencia del paradigma jerárquico, fue utilizado principalmente entre los años 1988 y 1992. Todavía es usado pero en menor medida desde 1992, ya que la tendencia fue a usar el paradigma hibrido. El paradigma reactivo surgió por el afán de los investigadores de inteligencia artificial en investigar la biología y sicología cognitiva, con el fin de examinar ejemplares vivos de inteligencia.
Otro de los motivos fue la rápida evolución de la tecnología y el abaratamiento del coste de ésta. Gracias a ello, permitió simular el comportamiento de una rana o insectos con un coste menor a los 500$, algo impensable en tiempos atrás.

El paradigma reactivo elimina por completo la planificación existente en el paradigma jerárquico, es una tipo de organización "percibir – actuar", como se muestra en la Figura 2.6

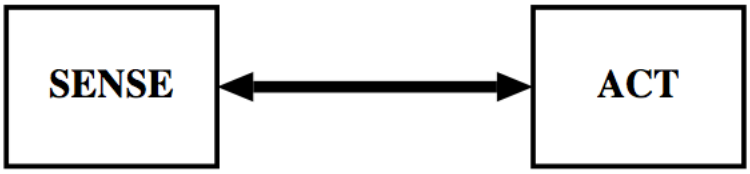

Figura 2.6. Organización del paradigma reactivo.

A diferencia del paradigma jerárquico, el cual asume que la entrada para realizar una acción es el resultado de una acción decidida por el planificador, en el paradigma reactivo la entrada para realizar una acción proviene de la información percibida de los diferentes sensores, se puede ver representado en la Figura 2.7.

El sensor esta directamente conectado a la acción a realizar, por ello el robot posee multitud de relaciones "información percibida – actuación". Estas relaciones son procesos concurrentes, denominados comportamientos, los cuales obtienen la información local del sensor y comprueban la mejor acción a realizar, independientemente de lo que están haciendo otros procesos. Por ejemplo, un

comportamiento puede dirigir un robot por medio de una acción de movimiento hacia delante para alcanzar una meta, mientras otro puede estar mandándole girar para evitar cierto obstáculo en su trayectoria. La actuación final es la combinación de todos los comportamientos.

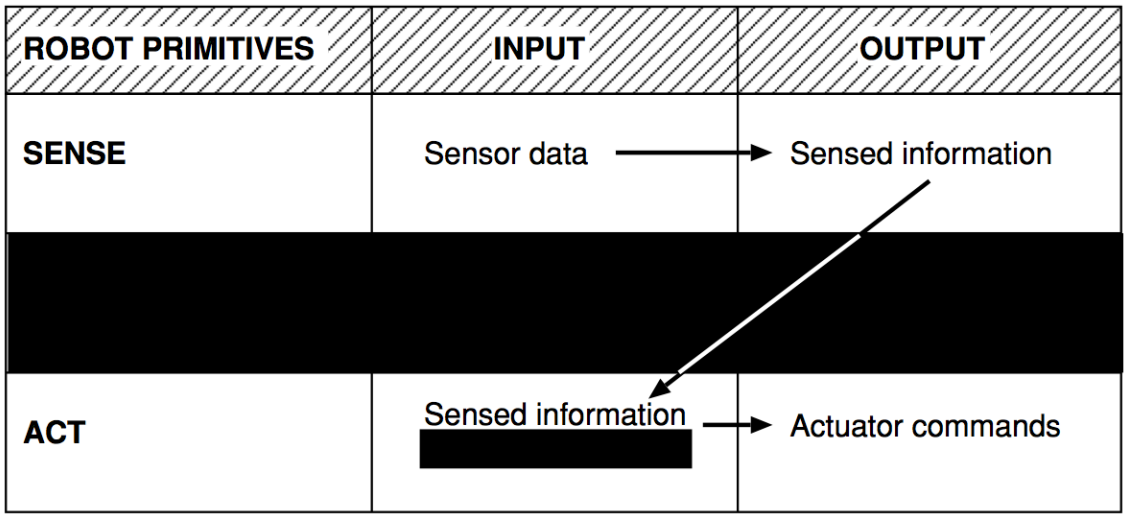

Figura 2.7. Flujo de actuación en el paradigma reactivo.

El sistema reactivo tiene muchas propiedades deseables, entre ellas está su rápido tiempo de ejecución, gracias a la eliminación del módulo de planificación. Esta eliminación es una opción demasiado extremista para robots de propósito general, pues la número de comportamientos que deberían tener para realizar sus tareas sería demasiado grande.

Es importante darse cuenta de que la inteligencia animal o natural no puede extrapolarse directamente a las necesidades y realidad de la programación de robots, pues una de las ventajas de la inteligencia animal sobre la robótica inteligente, es la evolución. Los animales evolucionan de una manera que les permiten la supervivencia de las especies, en cambio, los robots son caros y solo un pequeño número de ellos son construidos cada cierto tiempo.

2.4.3 Paradigma híbrido deliberativo/reactivo

El paradigma híbrido surgió en los años 90 y todavía hoy continua como área de investigación. En este paradigma, el robot primero planifica como descomponer una tarea mayor en diversas subtareas, dicha acción es realizada por el denominado planificador de misión (misión planning). Tras obtener éstas, el planificador de misión

determina cuales son los comportamientos adecuados para realizar dichas subtareas, tras ello, los comportamientos son ejecutados igual que en el paradigma reactivo.
La organización del paradigma hibrido, mostrado en la Figura 2.8, se descompone en dos etapas, en la primera se realiza la planificación y en la siguiente se realizan conjuntamente las tareas de percepción de información y actuación.

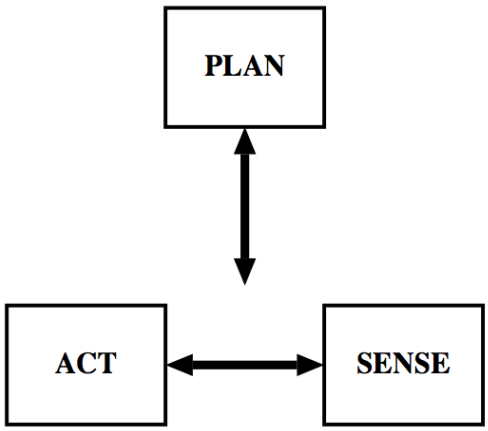

Figura 2.8. Organización del paradigma híbrido deliberativo/reactivo

El paradigma híbrido es una mezcla de los estilos reactivo y jerárquico. La información recibida de los diferentes sensores es enviada tanto al comportamiento, como al planificador, para que cree las subtareas basadas en un modelo del mundo global. Este planificador además puede escuchar de la información percibida por cada comportamiento, por ejemplo un comportamiento identifica un obstáculo, por lo que éste será modelado dentro en el mundo que construye el propio planificador, véase Figura 2.9.

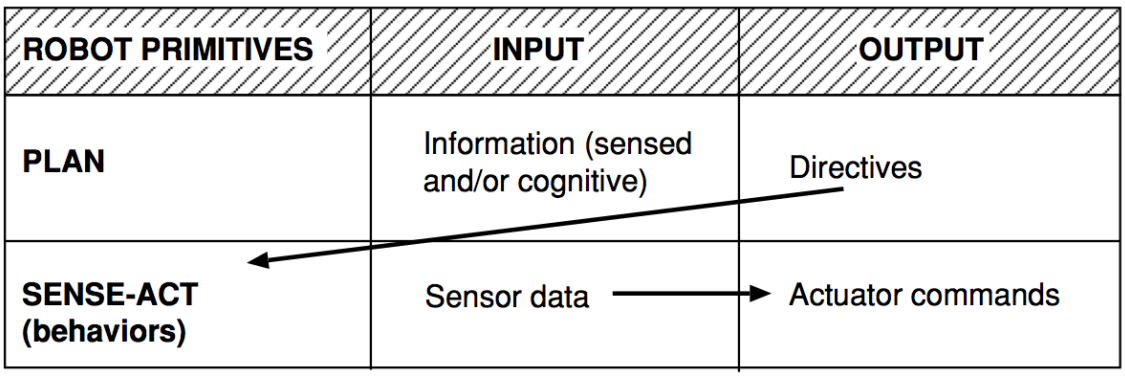

Figura 2.9. Flujo de actuación en el paradigma híbrido deliberativo/reactivo.

Este paradigma mejora a los dos anteriores, pues no se elimina el planificador como en el paradigma de reactivo y no solo se depende de éste para actuar, como en el paradigma jerárquico, además el tiempo de ejecución no supone un problema pues el planificador y los comportamientos ejecutan de manera independiente. Mientras que el planificador se actualiza cada 5 segundo, pues tiene una alto coste computacional, los comportamientos se ejecutan a un ratio de 0,16 segundos aproximadamente.

Todos los paradigmas y métodos de programación de robots son utilizados y mejorados por grandes grupos de investigación. Parte de el conocimiento o avances que consiguen son demostrados en grandes competiciones robóticas que existen alrededor del mundo.

A continuación se verán algunos de los grupos que dedican su estudio a la mejora de robots aéreos no tripulados y la competición internacional de robots aéreos denominada IARC(International Aerial Robotics Competition), donde estos colectivos de investigación muestran sus habilidades y avances, para llevar a cabo una misión específica.

2.5 Investigación y desarrollo en el campo de los UAV

Existen multitud de grupos de investigación alrededor del mundo que trabajan en el desarrollo de tecnologías autónomas para UAV. La mayor parte de estos grupos y sus logros se encuentra reflejados en las Figuras 2.10 y 2.11 (Kendoul, 2012). Están excluidos grupos de investigación militares o de grandes industrias .

Uno de los grandes retos para estos grupos de investigación no son solo de dotar con inteligencia al robot sino los medios con los que se les proporciona autonomía. Una de las funciones que se persigue es la navegación basada en visión, la cual es el principal objeto de estudio de uno de los grupos de investigación españoles, “Computer Vision Group” de la Universidad Politécnica de Madrid, y participes del proyecto en el que se basa el presente trabajo de fin de grado.

El Computer Vision Group (CVG) de la UPM posee una larga experiencia en el campo de la visión por computador. Su misión es estar a la última en cuanto a técnicas de control y procesado de imágenes con el fin de transferir dicha tecnología a aplicaciones civiles de los sistemas aéreos no tripulados.

Uno de los últimos logros cosechados por este grupo ha sido el primer puesto en la competición internacional de vuelo de microvehiculos aéreos (IMAV), en la categoría

NAME OF THE GROUP AND INSTITUTION	ROTORCRAFT PLATFORMS	CURRENT RESEARCH AREAS & PROJECTS	MILESTONES AND MAJOR ACHIEVEMENTS
Field Robotics Centre, CMU, CMU-FRC-US *www.frc.ri.cmu.edu/projects/landingdemo/* *www.frc.ri.cmu.edu/cademo*	**Class I:** Eurocopter EC135 and the Boeing unmanned Little Bird (ULB) helicopter **Class II:** Yamaha RMAX and R-50. **Class IV**: Otto Quadrotor	- Obstacles detection and avoidance using a 3D LIDAR. - Landing zone detection and safe landing using sweeping 2D LIDAR. - Visual localization and perception.	- Successful obstacle detection and avoidance with RMAX helicopter. - Safe landing area detection and automatic landing with full-scale helicopter .
UAV Research Facility, Georgia Institute of Technology, UAVRF-GTech-US *http://controls.ae.gatech.edu/wiki/uavrf*	**Class II**: Yamaha RMAX and SungWoo Eng. Remo-H helicopters. **Class IV**: Getpsy ducted-fan), GTQ (quadrotor) and GTLama (co-axial).	- Rotorcraft control using adaptive techniques and neural networks. - Vision-based navigation.	- Adaptive autopilot for 3D trajectory tracking and flight control. - Vision-based target tracking. - Vision-based formation flight .
The NASA/Army Autonomous Rotorcraft Project. NASA-ARP-US *http://ti.arc.nasa.gov/projects/apex/projectARP.php*	**Class II:** Yamaha RMAX helicopter.	- 3D navigation in urban environments using vision and LIDAR. - Safe landing area detection (PALACE project). - Mission and path planning.	- Mapping and path planning using a spinning LIDAR. - Vision-based state estimation. - Successful SLAD tests using stereo and LIDAR. - Mission planning for surveillance.
BEAR Group, Berkeley University, BEAR-US robotics.eecs.berkeley.edu/bear/	**Class II**: Yamaha RMAX and R-50 helicopters. **Class III**: Electric Maxi-Joker helicopters.	No current research activities on UAS (to our knowlwdge).	- Flight control using MPC. - Formation flight. - Vision-based landing. - LIDAR-based obstacle avoiadnce.
UASTech Lab., Linkopings University; UASTech-SE *www.ida.liu.se/divisions/aiics/aiicssite/uastech/*	**Class II**: Yamaha RMAX helicopter. **Class IV**: Quadrotors.	- Vision-based landing and localization. - Mission and path planning. - Artificial Intelligence	- Vision-based landing and localization - Path planning. - Mission planning and execustion monitoring.
The French Aerospace Labs (ONERA); ONERA-FR *http://action.onera.fr/accueil*	**Class II**: Yamaha RMAX helicopters.	- Vision-based navigation (target tracking and safe landing). - Mission planning and decisional autonomy (*ReSSAC project*). - UAS cooperation.	- Ground target tracking using vision. - Safe landing area detection using stereo vision. - Mission management.
USL, Univ. of South Florida; USL-USF-US *www.cse.usf.edu/USL*	**Class II:** Yamaha RMAX, **Class III**: Bergen, Raptor 90, Maxi Joker 2	- Flight control. - Vision-based navigation - Fault detection and isolation.	- Automatic flight. - Trafic data collection and analysis.
Kenzo Lab., Chiba University; CHIBA-U-JP *mec2.tm.chiba-u.jp/~nonami* *mec2.tm.chiba-u.jp/uav/main/*	**Class II**: Sky Surveyor, QTW, **Class III**: Hirobo SST-Eagle. **Class IV**: Astech quadrotors. **Class V**: Epson micro Flying Robot (uFR).	- Autopilots design. - Flight control. - Formation flight control. - Vision-based state estimation.	- Automatic flight of different rotorcraft platforms. - Vision-based flight. - Formation flight of two helicopters.
USW@ADFA, Australia *http://seit.unsw.adfa.edu.au/staff/sites/hrp/research/UAV/uav.html* *http://seit.unsw.adfa.edu.au/acme/researchconsult/flight/index.html*	**Class II:** Yamaha RMAX heli. **Class III**: Hirobo Eagle helicopter.	- Flight control. - Optic flow-based navigation. - Automatic landing on a ship.	- Optic flow-based terrain following using the Yamaha RMAX helicopter.
Shenyang Institute of Automation (SIA), SIA-CN *http://uav.sia.cn/en/index.php*	**Class II**: ServoHeli-120 (120 kg), ServoHeli-40 (40 kg).	- Avionics development/integration. - Flight control. - Path planning and UAS cooperation.	- Waypoint navigation with automatic take-off and landing. - Vision-based ground target tracking.
German Aerospace Centre (DLR); DLR-DE *http://www.dlr.de/ft/en/Desktopdefault.aspx/tabid--1377/1905_read-3350/*	**Class III**: ARTIS helicopter (25 kg)	- Vision-based navigation - Stereo vision-based mapping and obstacle avoidance. - Mission management.	- 3D mapping and path planning using stereo vision. - Vision-based flight through obstacle gates.
ARCAA (CSIRO-QUT), ARCAA-AU http://www.arcaa.aero/ http://research.ict.csiro.au/research/labs/autonomous--systems/field-robotics/field-robotics	**Class III**: Vario helicopters. **Class IV**: Quadrotors and Octocopters.	- Robust control of rotorcraft. - Dependable autonomous UAS. - Obstacle detection and path planning using vision and LIDAR.	- Beyond visual range (BVR) flights. - BVR infrastructure inspection. - Mapping and obstacle avoidance using LIDAR and/or stereo vision .
Aerospace Systems and Control Lab., KAIST, ASCL-KR *http://ascl.kaist.ac.kr/*	**Class III**: Voyager GSR 260 helicopter. **Class IV**: Electric T-REX 600 helicopter.	- UAS flight control. - Vision-based navigation. - Obstacle detection and path planning.	- Trajectory tracking and waypoint navigation. - Vision-based flight.
- National University of Singapore, NUS-SG *http://vlab.ee.nus.edu.sg/*	**Class III**: AF25B from Copterworks Inc; Raptor 90 **Class IV**: Trex-450 heli; coaxial rotorcraft.	- Nonlinear control of RUAS. - 3D indoor navigation. - Vision-based navigation.	- Automatic flight control. - Formation flight of a RUAS with a virtual leader. - Vision-based indoor flight.

Figura 2.10. Equipos de investigación UAVs..

NAME OF THE GROUP AND INSTITUTION	ROTORCRAFT PLATFORMS	CURRENT RESEARCH AREAS & PROJECTS	MILESTONES AND MAJOR ACHIEVEMENTS
- Computer Vision Group, Universidad Politécnica de Madrid, CVG-ES *http://www.vision4uav.com/*	**Class III**: Bergen Industrial Twin and Rotomotion SR20 heli. **Class IV**: Octocopter from Mikrokopter; quadrotor.	- Vision-based pose estimation. - Object tracking. - 3D mapping using vision. - Vision-based flight.	- Vision-based state estimation and ground target tracking. - Visual 3D SLAM. - Visual servoing.
- Robotics, Vision and Control Group , University of Seville, GRVC-ES *http://grvc.us.es/en/*	**Class III**: Heliv and HERO helicopters.	- Control of UAS. - Vision-based navigation and forest fire detection. - Cooperative perception and control of multiple UAS.	- Cooperative detection of forest fire using two RUAS. -Visual odometry and SLAM. - Cooperative faults detection.
- The Center for Advanced Aerospace Technologies (CATEC), CATEC-ES *http://www.catec.aero/*	fleet of heterogeneous UASs (about 6 fixed-wing UASs, 6 unmanned helicopters, and 10 quadrotors).	- Automatic flight control. - Navigation and sense & avoid. - Multi-vehicle coordination. - Avionics and onboard systems. - Automation and robotics.	- European COMETS and AWARE projects.
- Laboratory for Autonomous Flying Robots, TU Berlin, LFAFR-DE *http://pdv.cs.tu-berlin.de/lfafr/*	**Class III**: Aero-Tec CB-5000 heli. (12-16 kg). **Class IV**: Mikado Logo14, self-made quadrotor (5 kg).	- Control of VTOL UAS. - Collision detection & avoidance. - Distributed control of multiple RUAS.	- Load transportation using 3 helicopters. - Sensor nodes deployment by three autonomous helicopters.
- Autonomous Vehicles Group, Aalborg University, AVG-DK *http://www.es.aau.dk/projects/uav*	**Class III**: Bergen Industrial Twin, Vario XLC, Maxi Joker 2, T-Rex 700 Nitro, Thunder Tiger E550. **Class IV**: LMH-120 Corona, T-Rex 450, X-3D-BL quadrotor.	- Flight control with slung load. - Urban UAS navigation. - Task allocation and coordinated flight of AV groups.	- Control of a helicopter slung load system. - Robust control. - Collision-free path generation.
- Autonomous Helicopter project, *http://heli.stanford.edu/* - STARMAC project, *http://hybrid.stanford.edu/~starmac/project.htm*	**Class III**: 90-size XCell Tempest, 90-size Synergy N9. **Class IV:** STARMAC platform with quadrotors.	- Aerobatic and aggressive flight control. - Learning-based control. - Multi-agent control.	- Autorotation-based landing. - Learning-based aerobatic flight. - Bakflip control for a quadrotor. - collision avoidance flight.
- ACFR, University of Sysdney, ACFR-AU *http://www.acfr.usyd.edu.au/research/aerospace.shtml*	**Class III**: UAV vision G18 helicopter.	- Weed monitoring and animals tracking. - Multi-UAV active SLAM. - Cooperative UAS Systems	- Vision-based mapping and classification (fixed-wing). - Aquatic weed surveillance and management (helicopter).
-Aerospace Controls Lab, MIT, ACL-US. *http://acl.mit.edu/*	**Class IV**: Dragonfly quadrotors, AscTech quadrotors.	- UAS modeling and control. - Multi-UAS task assignement. - Multi-vehicle health management	- Indoor flight using VICON. - Persistent mission planning. - Health management of UAS.
- Robust Robotics Group, MIT, RRG-US *http://groups.csail.mit.edu/rrg/*	**Class IV**: AscTech quadrotors.	- Non-GPS indoor navigation. - Planning under uncertainties. - SLAM-based navigation and flight control.	- waypoint guidance and geolocation of ground targets. - Autonomous flight in unknown indoor environments. - Planning for target tracking.
-Autonomous Systems Lab., ETH, ASL-CH http://www.asl.ethz.ch/research/asl	**Class IV**: home-designed quadrotors and coaxial rotorcraft **Class V**: under-development for sFly project.	- AIRobots project (Inspection Rotorcrafts). - sFly project (Swarm of Micro Flying Robots).	- Design of miniature platforms. - Indoor hovering.
- Heudiasyc Lab, University of Technology of Compiegne, HDS-FR *http://www.hds.utc.fr*	**Class IV**: quadrotors and other home-designed multi-rotor configurations.	- Design of multi-rotor configurations. - Modeling and control of RUAS. - Vision-based navigation.	- Automatic hovering. - Vision-based hovering.
GRASP Lab., University of Pennsylvania,GRASP-US *http://alliance.seas.upenn.edu/~kumar/wiki/*	**Class IV:** AscTech quadrotors.	- Autonomous navigation and exploration. - Trajectory generation and control for 3D dynamic environments.	- Aggressive and accurate control of quadrotors using VICON system. - Cooperative manipulation and transportation with aerial robots.
CEA-USI-ANU Collaborative research between France (CEA and USI) and Australia (ANU)	**Class IV:** Home-built quadrotor.	- Nonlinear flight control. - Optic flow-based terrain following. - Visual servoing.	- Automatic flight. - vision-based hovering. - Terrain following in a structured indoor environment.

Figura 2.11. Continuación equipos de investigación en UAVs.

de autonomía de interiores. Las innovaciones mas sobresalientes fueron la completa autonomía de los UAV y la coordinación simultánea de varios de éstos. Esta categoría requiere que el vehículo vuele de manera autónoma en un escenario parcialmente conocido sin utilizar ningún dispositivo GPS, basándose únicamente en la navegación por visión.

El UAV debía atravesar una ventana y realizar diversas misiones de navegación en zonas con columnas que simulaban obstáculos. Cuanto mayor era el nivel de autonomía y el número de aparatos simultáneos en vuelo mayor era la puntuación. El CVG presentó una solución multirobot usando el AR-Drone 2.0, el cuál es una quadcopter fabricado por la compañía Parrot.

La evolución en el diseño y construcción de sistemas aéreos no tripulados, conseguida en gran parte gracias a la investigación de los citados grupos anteriores, se ve reflejada en los diferentes concursos internacionales de robótica existentes. Uno de los más importantes en el panorama de la robótica inteligente aérea y para el cual se basa este proyecto, es la competición internacional de robots aéreos "IARC".

2.6 IARC – International Aerial Robotics Competition

La primera misión de esta prestigiosa competición internacional, comenzó en 1991 en el campus del Instituto de Tecnología de Georgia. Es la competición internacional más longeva que todavía hoy en día se sigue celebrando. Desde su comienzo equipos universitarios, con el apoyo de la industria y el gobierno, han intentado completar con éxito cada una de las misiones propuestas en la competición, alcanzando comportamientos nunca antes conseguidos y avanzando así en el estado del arte de en la categoría de robots aéreos autónomos, lo cual es el principal objetivo del esta competición.

En los sucesivos años de la competición, se ha visto crecer las capacidades de éstos robots, desde vehículos que difícilmente se mantenían en el aire, hasta los más recientes vehículos que realizan una navegación totalmente autónoma y son capaces de interaccionar con el entorno.

Desde su origen, han pasado ya 23 años con un total de 6 misiones completadas. En 2013 se propuso la séptima y última misión hasta la fecha. A continuación se describen las diferentes misiones celebradas, y se entrará en mas detalle en las reglas de la séptima misión (IARC, 2014).

- **Misión 1:** La primera misión de todas consistía en transportar hasta seis discos metálicos situados aleatoriamente en la zona de recogida dentro de la arena, llevarlos hasta el otro lado de una barrera central con una altura de 3 pies(1m aproximadamente) y soltarlos en la zona objetivo. Durante los dos años siguientes al inicio de esta misión, las diferentes universidades participantes continuaron mejorando sus vehículos, hasta que el equipo del Instituto de tecnología de Georgia, consiguió realizar el primer vuelo autónomo, en el que se realizaban automáticamente las fases de despegue, vuelo, y aterrizaje. Tres años después en 1995 el equipo de la Universidad de Stanford consiguió llevar desde un lado de la arena al otro uno de los discos. Como la misión consistía en una consecución de puntos, el que más discos llevase al otro lado mayor puntuación obtenía, como en este caso el único que pudo transportar un disco fue el equipo universitario de Stanford, se les adjudicó la victoria.

- **Misión 2:** Esta segunda misión nada tenía que ver con la primera, requería que los equipos buscasen una serie de bidones con residuos tóxicos, localizando su posición con unas coordenadas GPS, identificar el contenido de cada bidón por medio de una etiqueta de materiales peligrosos que se encontraba en el exterior de cada bidón y por ultimo obtener una muestra de uno de los bidones.

  En 1996 un equipo de Instituto tecnológico de Massachusetts y la universidad de Boston, con el apoyo de los laboratorios Draper, crearon un vehículo totalmente autónomo que reconocía la localización del total de cinco bidones con residuos tóxicos e identificaron el contenido de dos de ellos desde el air, pero si ni siquiera intentar la recogida de muestra. El año siguiente, en 1997 el equipo de la Universidad de Carnegie Mellon, consiguieron identificar la posición y el contenido de todos los bidones existentes en la competición, pero cuando intentaban recoger la muestra, fallaba perdiendo el objetivo por unos pocos centímetros. Finalmente se le otorgó la victoria al equipo de Carnegie Mellon.

- **Misión 3:** Iniciada en 1998, se trataba de una misión de búsqueda y rescate. Los robots debían despegar y volar a una zona devastada por un desastre natural y buscar supervivientes, entre cuerpos sin vida quemados, cañerías de agua rotas, nubes de gas toxico y edificios derrumbados. En el año 2000 un robot construido por el equipo de la Universidad Técnica de Berlín fue capaz de detectar y evitar todos los obstáculos descritos, además de identificar y diferenciar entre los supervivientes y los cuerpos sin vida, transmitiendo imágenes de los supervivientes junto con su localización para los primeros rescatadores en llegar a la zona.

- **Misión 4:** Iniciada en 2001, contemplaba tres escenarios distintos en los que el comportamiento del robot aéreo debía de ser el mismo. El primero era una misión de rescate de rehenes en una embajada de un país del tercer mundo, debían entrar en dicha embajada y tomar fotografías de los rehenes y de sus captores. En el segundo escenario debían encontrar un mausoleo arqueológico donde habían fallecido unos arqueólogos, entrar dentro y tomar ciertas fotografías antes de su destrucción. En el último escenario, se plantea una explosión en una central nuclear, la cuál había matado a todos los operarios, el robot debía encontrar el edificio de operaciones de la central, entrar en éste y tomar fotografías de los paneles de control para saber si se iba a producir inminentemente una fusión del núcleo. Estos tres escenarios tenían en común las mismos objetivos:

    - Partir desde una localización a 3km de distancia del edificio objetivo.
    - Localización del complejo.
    - Localización de un edificio en concreto del complejo.
    - Identificación de las posibles entradas al edificio.
    - Entrar por dichas posibles aperturas.
    - Volver al punto de partida tras hacer fotografías del objetivo.
    - Tiempo limitado para la misión de 15 minutos.
    - Autonomía total en todos los aspectos de la misión.

  La misión se dio por completada en 2008, todos los equipos fueron capaces de demostrar el comportamiento demandado por medio de los diferentes objetivos, a excepción de cumplimentarlo en el tiempo previsto de 15minutos. Por lo que el jurado decidió repartir el premio de la misión 80.000$ entre los participantes asignándoles a cada grupo una cantidad basándose en el rendimiento demostrado.

- **Misión 5:** Esta misión era parte de uno de los escenarios de la cuarta misión fue lanzada en 2009. En ella se debía penetrar dentro de un edificio y sortear una serie de dificultades mayores que en la cuarta misión. Entre estos se encontraban, un menor espacio permisible para el movimiento, pasillos sin salida, encontrar el objetivo sin la ayuda de GPS, tomar fotografías de éste y volver al punto de partida a cierta distancia del edificio.

  Fue la primera vez que se consiguió en el mismo año que se planteaba la misión. Los ganadores fueron el equipo de investigación del Instituto tecnológico de Massachusetts(MIT). Realizaron un vehículo autónomo que se utilizaba un

sistema laser para crear un mapa de su entorno, así como un sistema óptico que le ayudaba en su posicionamiento y movimiento. Hasta poco antes de que terminara su cuarto intento el robot del MIT consiguió identificar el objetivo, fotografiarlo y enviar las fotos al jurado.

- **Misión 6:** Propuesta en 2010, esta misión era una extensión de la quinta misión, pero requería unos comportamientos por parte de los vehículos mucho más avanzados, comportamientos que nunca antes habían sido demostrados. El escenario era una misión de espionaje en el que el robot aéreo debía robar un pen-drive, el cuál se encontraba en una habitación de la planta de un edificio, de la que se desconocía su plano. Tras localizar el pen-drive debía cogerlo, depositar otro pen-drive idéntico para evitar la detección del robo y devolver el original en una localización específica.

  No fue hasta el cuarto año cuando el equipo de investigación de la universidad de Tsinghua de China consiguió completar la misión. Utilizaron un quadcopter UAV que lograba localizarse con una precisión de centímetros, gracias al desarrollo de algoritmos basados en visión 3D, todo ello sin ayuda de posicionamiento GPS. Recibieron un premio total de 40000$, pues se había acumulado 10000$ por año de misión.

2.6.1 Misión 7

En Octubre de 2013 se propuso la séptima y última misión de la competición hasta el momento. En esta misión los equipos deberán demostrar la capacidad de rastrear otros robots autónomos de tierra que se mueven aleatoriamente e interactuar con ellos, además de conseguir una navegación autónoma sin localización GPS. Por ello uno de los comportamientos que se buscan en esta misión y que todavía esta en su nacimiento dentro de la robótica inteligente, es el "sentir y evitar". Los vehículos deberán sentir la presencia de objetos fijos y en movimiento y evitar su colisión.

Esta misión se dividirá en dos submisiones "Misión 7a" y "Misión 7b". En la Misión 7a los concursantes deberán demostrar los comportamientos arriba descritos, mediante la cumplimentación de una serie de objetivos que más adelante se describirán. En la Misión 7b, será una competición cara a cara entre dos equipos en la que se realicen los mismos pruebas, pero en este caso el que logre más objetivos que el equipo contrario ganará.

A continuación se encuentra la descripción general de la prueba.

Esta se desarrollará en un cuadrilátero denominado “Arena” de 20 metros de lado. Delimitado por líneas de 8 centímetros de grosor con una anchura de diferentes colores como se muestran en la Figura 2.12.

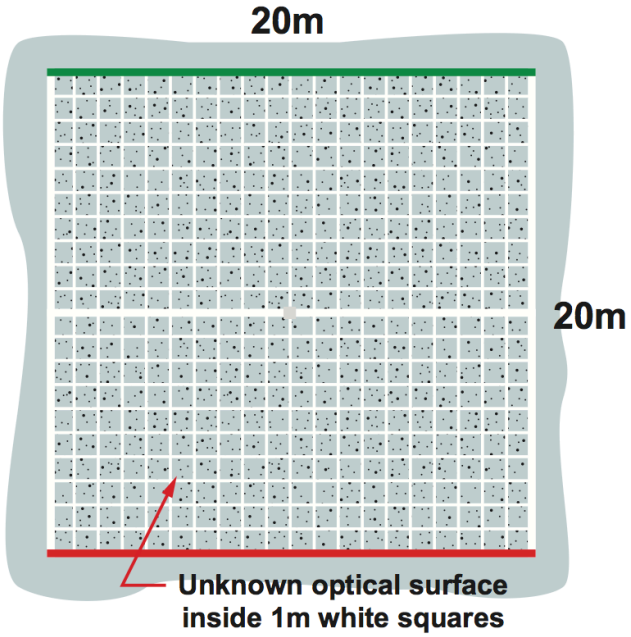

Figura 2.12. Arena de la competición.

Dentro de la Arena habrá 10 robots de tierra autónomos modelo “iRobot Create” .La posición inicial de estos será en el centro de la arena dispuestos de tal manera que puedan moverse hacia todas las direcciones de la arena.

El comportamiento de los robots de tierra es el siguiente. Una vez comienza la prueba empezaran a moverse hacia los limites de la arena, transcurridos 20 segundos o en caso de que ocurriese una colisión el robot cambiará su dirección 180º en el sentido de las agujas del reloj. Además, cada 5 segundos el robot variaría aleatoriamente su dirección en un rango de entre 0º y 20º. Cuando alguno de los robots cruza una de las líneas límites de la arena, será eliminado de la misión.

Cada uno de los robots de tierra posee en la parte superior un sensor magnético, cuando el UAV se acerque lo suficiente como para que el de tierra lo detecte, éste cambiará su dirección 45º en el sentido de las agujas del reloj. Se deberá tener en cuenta que dos descensos seguidos sobre el mismo robot 90º, es decir por cada descenso 45º de giro. Además si el robot desciende con el fin de cortar la trayectoria del robot de tierra, haciendo que éste colisione con él, el robot de tierra girará 180º en las agujas del reloj.

Además de los 10 robots de tierra, existirán dentro de la arena otros 4 robots, denominados robots obstáculos. Éstos robots serán del mismo modelo que los de tierra pero físicamente tendrán en la parte superior un cilindro que se extiende verticalmente hasta una altura de 2 metros como máximo. El comportamiento de estos robots será dar vueltas en círculos alrededor de la arena, en el sentido de las agujas del reloj. Tienen dos misiones, la primera será la de crear colisiones con los 10 robots de tierra, para hacer el movimiento de éstos más aleatorio, y segundo servir como obstáculo verticales que deben ser evitados por el UAV, con el fin de que éste use el comportamiento buscado de "sentir y evitar". En caso de que el UAV chocase con algún obstáculo, la prueba se dará por finalizada.

El objetivo del UAV será analizar las direcciones de los robots de tierra, teniendo en cuenta el comportamiento de éstos, y posteriormente redirigir o pastorear todos los robots de tierra hacia la zona de la arena delimitada con una línea verde, a la vez que evita los robots obstáculo. El UAV deberá permanecer dentro de la arena, aunque se permite que exceda estos limites hasta dos metros y durante un tiempo no superior a los 5 segundos. Como limite superior no podrá exceder los tres metros de altura sobre el suelo, pero si se le permite que aterrice dentro de la arena.

La prueba finalizará cuando una de las siguientes condiciones se cumpla, todos los robots hayan traspasado la línea verde, o cualquiera de las líneas limitantes de la arena, o cuando tras 10 minutos y todavía haya robots de tierra que no hayan pasado la línea verde se considerarán como que han abandonado la arena para términos de puntuación. Completar la prueba en un tiempo inferior a los 10 minutos será un factor determinante para la selección del ganador del premio.

Para paliar los efectos de la mala suerte, cada equipo tendrá un total de 3 intentos y aquel en que se obtengan los mejores resultados, será el utilizado para puntuar al equipo participantes. El equipo con la mayor puntuación será el ganador de la misión 7a.

Esta misión ha sido diseñada para dominar diferentes capacidades y estrategias como son la velocidad, autonomía, reconocimiento de objetos, interacción entre robots, seguimiento de robots, control en la maniobra de aterrizaje y descenso hacia un objetivo, identificación y priorización de objetivos, conocimiento del entorno y del progreso de la misión. Todo esto deberá ser demostrado en la misión 7a con el fin de poder pasar a la misión 7b, en la que dos equipos se enfrentarán cara a cara y el que consiga llevar más robots hacia su zona ganará.

2.7 Equipamiento del vehículo aéreo para la competición

Para afrontar la competición antes descrita, se usará como plataforma el AscTec Pelican Quadrotor fabricado por Ascending Technologies, se muestra en la Figura 2.13. La estructura del vehículo, motores y hélices se usaran sin realizar modificación alguna, sin embargo, se fabricará un nuevo tren de aterrizaje junto con las monturas para cámaras.

El Autopilot AscTec, computador montado a bordo del robot, es un herramienta de investigación precisa y multifuncional, pude verse en la Figura 2.13. Esta unidad de control del vuelo, contiene todos los sensores necesarios para funcionar como una unidad de medición inercial, además posee dos microprocesadores ARM7 y diversos interfaces de comunicación.

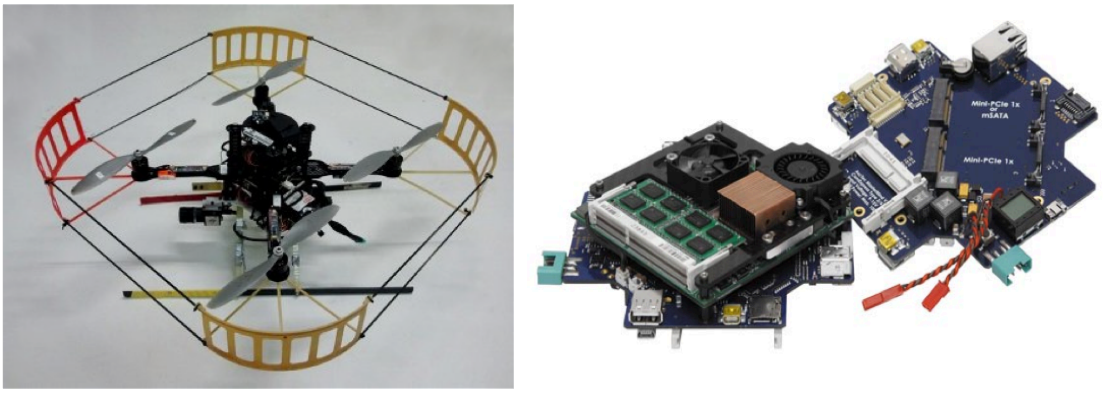

Figura 2.13. (Izq.) AsTec Pelican Quadrotor. (Dcha.) Computador de abordo AscTec Mastermind.

Para conseguir dotar al vehículo mostrado con todas las características que se han descrito como objetivo en la misión 7, durante la construcción y diseño del UAV hay que lidiar con la problemática de realizar el uso de éste en un ambiente no estructurado y complejo, como el requerido en dicha misión. Además, de otras dificultades como el alto coste, la peligrosidad o incluso la imposibilidad de llevar el proyecto a cabo.

Un único agente puede tener un conjunto de comportamientos pequeño, pero cuando un grupo de agentes actúan conjuntamente, la apariencia es de un sistema complejo que evoluciona con el tiempo. En consecuencia de esto, surge la necesidad de realizar una simulación del robot y su entorno antes de iniciar su construcción en el mundo real. Aunque apenas se trate de una simulación de lo que ocurre en el mundo abierto, las experiencias del juego permitirán observar las dificultades de los ambientes complejos y las decisiones que debe tomar el ingeniero para resolver las mismas.

Todo esto se ve reflejado en el trabajo expuesto en la presente memoria, pues para llegar al objetivo de obtener una arquitectura de control, es necesario la construcción de un entorno simulado en el que diferentes agentes interactúan entre sí y cuyo principal actor es un robot aéreo.

Este simulador ayudará principalmente a dos cosas, la primera, será la de crear una arquitectura de control para dotar al propio robot de inteligencia y ver como este interacciona con el resto de los agentes en ese entorno simulado y por último, elegir la mejor estrategia a seguir por el UAV, es decir, el mejor mission planner.

3. DISEÑO E IMPLEMENTACIÓN

Este capítulo muestra la aplicación del paradigma hibrido para el diseño de la arquitectura software. También presenta el sistema software de robótica ROS y la aplicación de éste para la implementación de los módulos que forman el conjunto de la arquitectura.

3.1 Aplicación del paradigma híbrido deliberativo/reactivo

Para el diseño de esta arquitectura software se deberán afrontar diversos problemas o requerimientos. La principal y más importante característica requerida, será la modularidad. Es necesario la utilización de distintos módulos que permitan mejorarlos o modificarlos sin afectar al funcionamiento del resto de módulos.

El modelo que permite la modularidad deseada, es el modelo GNC (Guidance Navigation and Control), el cual se ha visto en el marco de clasificación de autonomía ALFURS. Además de éstos tres módulos, será necesario otro más para llevar a cabo la simulación y prueba de el resto de componentes. Por consiguiente, esta arquitectura estará formada por los siguientes módulos, el "Flight Control System", "Guidance system" que será denominado a partir de ahora "Mission Planner", pues para este trabajo se ha considerado que es más propio, ya que es el único encargado del tomar las decisiones de guiado del UAV, el "Navigation System" que llevará acabo las funciones de recrear un modelo del mundo, que en este caso será un mapa de la arena del concurso con los diferentes agentes que el UAV perciba, y por ultimo el módulo de simulación "Simulation system".

Para que esta arquitectura funcione es necesario que haya una interconexión entre los distintos módulos. El Navigation system percibirá el entorno, comunicándoselo al misión planner que tomara una decisión de actuación, y al flight control system que mandará ejecutar dicha acción basándose también en la información de su entorno. Más adelante se verá con más detalle como funcionan cada una de estas conexiones. Se puede observar en la Figura 3.1, la construcción de la arquitectura y el flujo de comunicación existente entre cada uno de los módulos.

Es posible identificar la funcionalidad de esta arquitectura, con el flujo de actuación del paradigma híbrido deliberativo/reactivo, pues es el elegido para ser usado como método de programación de esta arquitectura software. Este paradigma es necesario, por el tipo de comportamiento autónomo que deberá tener el UAV. No sólo se necesita planificar la actuación sobre los diferentes robots de tierra, sino que también es necesario unos

comportamientos reactivos a ciertos elementos, como el esquivar robots obstáculos, impedir que el UAV abandone los limites de la arena o evitar que los robots objetivo estén cerca de los limites no deseados.

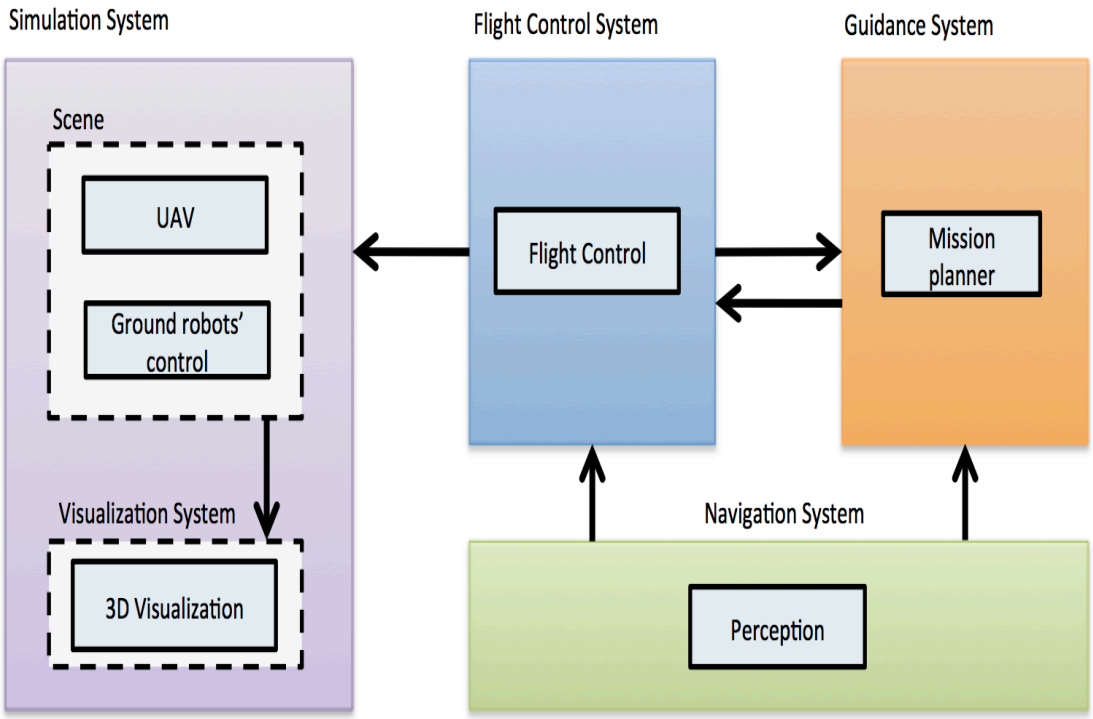

Figura 3.1. Diseño abstracto de la arquitectura software.

La relación del paradigma hibrido con la solución adoptada para la construcción de la arquitectura software es la siguiente:

- "PLAN" → Mission planner.
- "SENSE" → Navigation system.
- "ACT" → Flight Control system.

La función de planificador del paradigma híbrido será llevada a cabo por el misión planner. Éste tomará una decisión, dependiendo de los datos recogidos por el modulo de navegación. La decisión es enviada al modulo flight control system para que la ejecute. Además de ejecutar la acción, sirviéndose de los datos del mundo recogidos por el Navigation system, tendrá un determinado comportamiento.

Se puede ver la aplicación del paradigma híbrido a la solución adoptada para la construcción de la arquitectura en la Figura 3.2.

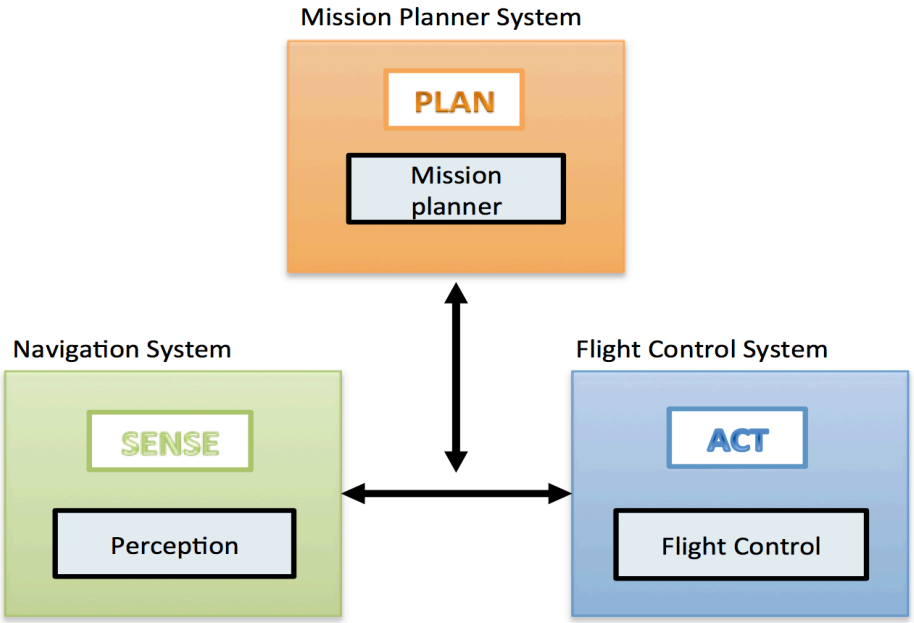

Figura 3.2. Arquitectura basada en el paradigma híbrido

Una vez decidida la estructura de la arquitectura, es necesario unas herramientas software y un lenguaje de programación que nos permitan implementarla. Tras realizar una análisis de sistemas software, se llego a la conclusión de que el software que proporcionaba las mejores herramientas para construir una arquitectura modular y visualizar el comportamiento de ésta, era ROS(Robotic Operating System).

3.2 ROS – Robotic Operating System

Es un conjunto de herramientas software o framework que permite desarrollar software de robótica, aportando todo el potencial como el de un sistema operativo. Oferce una gran variedad de herramientas de manera modular, es decir, es posible elegir hasta que punto se desea usar ROS. Las herramientas que conforman el denominado “Ecosistema ROS” están separadas en tres grupos distintos:

- Herramientas independientes del lenguaje o plataforma para desarrollar software, como Rviz o Rqt.

- Implementaciones de la librería cliente de ROS en dos lenguajes principalmente, Roscpp(C++) y Rospy (Python).
- Paquetes desarrollados con dichas librerías por otros usuarios y que podrían servir de ayuda a nuestras implementaciones.

Las características mas importantes de este framework son las que figuran a continuación (ROS 2007):

- **Una detallada infraestructura de comunicación.** Gracias a su construcción nos proporciona una interfaz de comunicación entre procesos denominados nodos ROS, abstrayéndonos de su implementación y funcionando así como una capa middleware. Esta interfaz de comunicación nos permite:

    - *Paso de Mensajes:* Posee un sistema de paso de mensajes por medio de mecanismos de editor-subscriptor. Cada uno de los nodos ROS podrá escribir en un tema específico y a su vez escuchar de los temas que requiera, recibiendo los mensajes que se publiquen de dicho tema.

    - *Llamadas a procedimientos remotos:* Además de utilizar mecanismos de editor subscriptor. Se pueden crear servicios remotos, que realicen determinada acción cuando éstos son invocados.

- **Características específicas de robótica.** Posee determinadas características que nos permitan desarrollar software de robótica de una manera más rápida, al estar este ya adecuado para robots.

    - *Mensajes estándar de robots:* Existen gran variedad de mensajes, desde los más básicos para enviar un número entero o en coma flotante hasta mensajes complejos de conceptos geométricos como son la actitud o la posición. Incluso se pueden crear nuevos mensajes, a partir de los ya existentes.

    - *Librería de geometría de robots:* Uno de los principales problemas en los robots humanoides, es saber donde se encuentran y como se encajan cada uno de sus partes o componentes. Ros no ayuda gracias a su librería "TF" (transform). Usada para gestionar datos de transformaciones de coordenadas con más de cien grados de libertad y actualizar a una tasa de cientos de Hz. Nos permite definir transformaciones estáticas, como una cámara fija a una base móvil y transformaciones dinámicas, como una articulación de un brazo de robot.

- *Lenguaje de descripción de robot*: Gracias a ROS es posible describir el robot de una manera entendible por la propia máquina. Éste proporciona una serie de herramientas que nos permita modelar nuestro robot de una manera que sea entendible por el resto de componentes de ROS. El formato para describir el robot es URDF(Unified Robot Description Format) el cual consiste en un documento XML que describe la propiedades físicas del robot.

- **Variedad de herramientas.** Las herramienta que nos proporciona ROS sirven de apoyo para la depuración, trazado y visualización del sistema desarrollado. Para un modelo basado en editor-subscriptor éstas herramientas nos permiten escuchar los datos que fluyen por el sistema, de una manera espontanea. Las herramientas de ROS se sirven de ésta introspección a través de una extensa colección de utilidades de líneas de comandos y gráficos que simplifican el desarrollo y la depuración, como:

  - *Rviz:* Una de las herramientas más conocidas de ROS, ya que nos permite visualizar en 3D la información recibida por los distintos sensores del robot y el propio robot descrito en el formato URDF. Esta visualización es posible gracias al paso de mensajes existente, Rviz utiliza cualquiera de los mensajes estándar proporcionados para mostrar de manera tridimensional, cualquier información percibida por el robot. Esta visualización nos permite ver rápidamente lo que el robot percibe e identificar problemas en el diseño de éste.

  - *RQT:* Es un framework basado en la librería QT que nos permite realizar interfaces gráficas. Este engloba diversos plugins, uno de los más importantes y que se utilizará más adelante, es “rqt_graph”, el cual se sirve de la introspección de ROS para mostrar gráficamente las conexiones entre nodos dentro de una arquitectura basada en el modelo de editor-subscriptor. Además, para estos modelos basados en editores-subscriptores existen otros plugins de depuración como “rqt_topic”, para escuchar de un tema en concreto o “rqt_publisher”, para publicar mensajes en ciertos temas. Otro de los plugins v es “rqt_plot” que nos permite visualizar encoders, voltajes o cualquier valor que varíe con el tiempo

Sirviéndonos de todas estas herramientas, se puede crear un arquitectura de control de manera modular, en la que cada uno de los módulos de la arquitectura será un nodo ROS y la comunicación de éstos se hará a través de un sistema de editores-subscriptores.

Un nodo ROS, tiene el mismo funcionamiento que un proceso, el cuál tendrá una función denominada "Run" que estará en un bucle continuo de ejecución hasta que el proceso muera o en nuestro caso, hasta que el servicio ROS deje de ejecutar. Esté nodo puede alterar su modo de ejecución por el medio del paso de mensajes síncrono, ya que cuando éstos se reciben se produce la llamada a una función determinada. Cada uno de los nodos publicará y recibirá mensajes de los temas que necesite para su funcionamiento. Todos éstos pasos de mensajes se realizan a través del nodo maestro denominado "ROS Core", pues es el que proporciona el servicio.

Para la visualización de la simulación se utiliza Rviz, de manera que permita ver como es el comportamiento del UAV y su interacción con el resto de agentes, para así realizar la depuración y mejora de éste.

Existen distintas distribuciones de ROS, las ya descontinuadas "Box Turtle", "C Turtle", "Diamondback", "Electric Emys" y "Fuerte Turtle", y las actuales distribuciones "ROS Groovy Galapagos" y "ROS Hydro Medusa", que es la recomendada y más actual y que se utiliza en este proyecto, junto con la distribución Linux Ubuntu 12.04.

Para la programación del robot se usara el lenguaje de programación Python pues es uno de los dos lenguajes para los que existe la librería cliente de ROS, además de que éste es un lenguaje de alto nivel con una sintaxis muy limpia, que permite obtener un código fácil de leer, multiplataforma, con soporte para orientación a objetos, programación imperativa y programación funcional. Existen multitud de paquetes desarrollados en ROS con Python, así como ejemplos sencillos a modo de tutorial en la página web oficial de ROS (ROS 2007), que pueden servir como guía para aprender su funcionamiento. Además, existe un foro dentro de la propia pagina de ROS y su correspondiente wiki, que servirá de ayuda, mediante la realización de preguntas y por medio de las soluciones dadas a los problemas surgidos por otros usuarios.

Una vez vistas las diferentes herramientas que se tienen para la implementación, se muestra la funcionalidad, diseño e implementación de cada uno de los distintos módulos que conforman la arquitectura. Antes de ello es necesario conocer los mensajes existentes que se intercambian entre los diferentes módulos, con el fin de tener un entendimiento mayor de su funcionamiento.

3.3 Mensajes internos de la Arquitectura de Control

Los mensajes utilizados, son mensajes construidos a partir de los mensajes estándar proporcionados por ROS y por otros mensajes construidos por el propio, éstos mensajes son los siguientes:

- **Decision.** Mensaje utilizado para enviar la decisión del Mission planner. Este contiene los siguientes elementos:

    - *Action (uint16).* Será la acción a ejecutar por el Flight control system. Puede tomar los valores TOUCH = 0 o CLOSE = 1.

    - *Robot_number (int16).* El número de robot sobre el que debe actuar el UAV.

    - *Touch_times (int16).* En caso de que la acción sea tocar a un robot, este campo llevara asociado el número de veces que debe ser tocado dicho robot.

- **FlightControlState.** Este tipo de mensaje será utilizado para enviar el estado del flight control. Está formado por la variable:

    - *Control_state (uint16):* Esta variable podrá tomar dos valores FREE = 0 o BUSY = 1, dependiendo de

- **Map.** El mensaje será utilizado para enviar la información percibida por el modulo del Navigation system, así como el estado de todos los agentes que intervienen en la simulación, para su visualización. Este mensaje esta compuesto por un mensaje estándar de ROS y un mensaje que específicamente se ha construido para almacenar el estado de los robots. Los elementos que contiene el mensaje Map son los siguientes:

    - *DronePosition (*geometry_msg/Point)*:* En este elemento se almacena la posición del UAV. Esta variable será del tipo "geometry_msg/Point", la cuál almacena los valores de las coordenadas cartesianas de la posición, por lo que posee los campos "X" e "Y".

    - *Targets (GroundRobotsState[]):* Lista que contendrá el estado de los diferentes robots objetivos de la misión. Estos estados se almacenan en una lista de un tipo de mensaje personalizado, que se ha denominado

"GroundRobotsState". Este tipo de mensaje a su vez, contiene los siguientes elementos:

- Position(geometry_msg/Point). En esta variable se almacenará la posición del robot. Se almacenan las coordenadas cartesianas en un mensaje estándar de ROS, descrito anteriormente.

- Direction(float64).Variable que contendrá la dirección en grados del robot.

- State (uint16). Contiene el estado actual del robot. Esta puede tomar los siguientes valores RUN = 0, TOUCHED = 1, NOISE = 2, REVERSE = 3, DELETED = 4.

- Last_reverse (float64). Es el tiempo que ha pasado desde que el robot realizó el ultimo cambio de dirección de 180º, el cual se produce cada 20 segundos.

- *Obstacles (GroundRobotsState[]):* Almacena el estado de los robots obstáculos. Al igual que el anterior se usa una lista del tipo de mensaje "GroundRobotsState", aunque algunos de las variables de éste no se utilicen para los robots obstáculo como "state" o "last_reverse".

Tras haber visto todos los mensajes que se pueden intercambiar los diferentes módulos, es posible entender mejor como funciona la comunicación entre los módulos, los cuales se describen a continuación.

3.4 Mission Planner

Será el encargado de decidir la siguiente acción a ejecutar por el módulo flight control, sirviéndose de la información recogida en el módulo de navegación.

El mission planner genera metas a largo plazo para que sean alcanzadas en los siguientes 5 o 10 segundos. Se ha diseñado un mission planner siguiendo una estrategia que ha sido denominada "seleccionar y guiar". El UAV primero "selecciona" un robot objetivo apropiado, teniendo en cuenta ciertas características (distancia entre el UAV y los robots objetivo, distancia a la línea de meta, etc…). Posteriormente, el UAV guía el robot seleccionado hasta la línea de meta, interaccionando con el robot en momentos específicos. Durante la ejecución de esta estrategia, el proceso puede fallar debido a

ciertas situaciones, como por ejemplo, que haya obstáculos en su trayectoria. En este caso, el UAV puede decidir abandonar el robot seleccionado y elegir otro distinto.

El mission planner usa como datos de entrada el modelo mundo observado, que es la información recogida por el módulo de navegación. Ésta incluye, por cada obstáculo, su localización espacial y por cada robot objetivo percibido, su localización espacial, velocidad, orientación y tiempo de ciclo(tiempo para el siguiente fin de ciclo de los 20 segundos). El mission planner genera acciones, es decir, metas especificas asociadas a comportamientos que pueden ser entendidos y alcanzados por el módulo del flight control system. En cada ciclo, el mission planner envía una acción específica al flight control system para que ésta sea ejecutada. Una vez es enviada esta acción, el mission planner se bloqueará hasta que la acción es llevada a cabo por el flight control. Cuando este queda libre, solicitará más acciones al mission planner.

Se ha implementado el mission planner como un planificador jerárquico. La Figura 3.3 muestra una vista parcial de la descomposición jerárquica de metas en sub-metas. La meta que esta en lo mas alto de la jerarquía, "HerdRobots", representa el objetivo general del UAV, éste es el de guiar un conjunto de robots hasta la línea de meta. Puede ser realizada por un conjunto de metas mas simples del tipo "Guide", la cual guía un robot especifico localizado en una posición dada, hasta la línea de meta. Esta meta puede ser descompuesta a su vez en metas de un nivel más bajo que ya son las que ejecuta el flight control y que se verán más adelante, "GetClose" y "Touch". Este es un proceso recursivo, ya que a un nivel bajo puede haber metas que también se presentan a niveles más altos de la jerarquía.

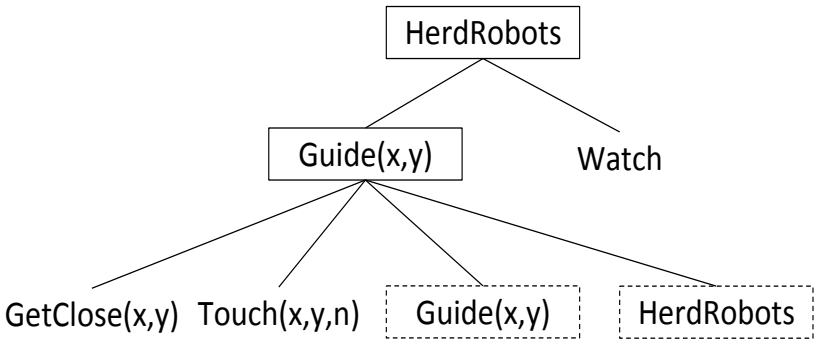

Figura 3.3. Comunicaciones del nodo "Mission Planner".

Este módulo es implementado como un nodo ROS que toma el nombre "missionPlanner" dentro de la arquitectura formada en el sistema ROS. Este nodo necesita tener comunicación con los nodos del Flight control system y el Navigation

system por medio de los diferentes topics o temas, representados en la Figura 3.4, éstos son:

- **Decision.** Este nodo tiene un editor de mensajes, por el que enviar la acción decidida a ejecutar. El tipo de mensaje a enviar será del tipo "Decision".

- **Control_state.** Se realiza una subscripción a dicho tema, para obtener el estado en todo momento del Flight control system, por si ha acabado de ejecutar la anterior acción y está libre, poder mandarle la siguiente acción. Una vez llega el mensaje de que este esta libre, se ejecuta una función que pondrá a trabajar al mission planner. Los mensajes de dicho topic son del tipo "FlightControlState".

- **Map_state.** Con la subscripción a este topic se realiza la comunicación con el nodo del Navigation system, pues por él se recibe la información del mundo percibida a través de los diferentes sensores de dicho módulo. El mensaje que se recibe es del tipo "Map".

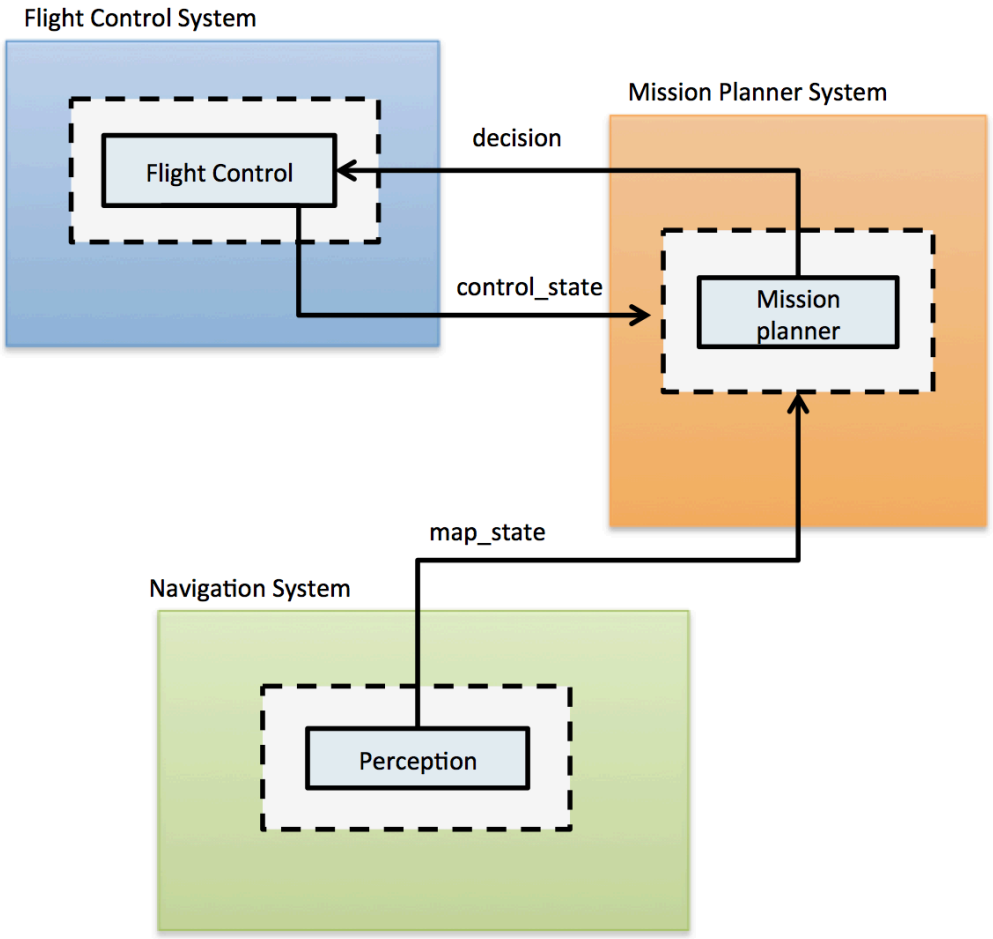

Figura 3.4. Comunicaciones del nodo "Mission Planner".

3.5 Navigation System

Este módulo es encargado de recoger la información percibida por los diferentes sensores del UAV, y posteriormente enviarla a los módulos del mission planner y el flight control system.

El diseño de éste será muy básico pues es el equipo de visión quien se encargue de implementarlo. Para las pruebas necesarias únicamente recogerá los datos generados en el modulo de simulación, en particular del submódulo que se encarga de generar la escena dentro de la arena, y lo transmitirá a los módulos pertinentes.

Al igual que el resto de módulos, éste será implementado como un nodo ROS, al que se llamará "perception". El nodo perception estará conectado con el resto de nodos que son el flight control, mission planner y simulation. Esta comunicación se hará a través de diferentes topics que se representan en la Figura 3.5, estos son:

- **Ground_robots_state.** Se realizará una subscripción a dicho topic para recibir del nodo de simulación el estado de los diferentes robots de tierra, es decir, los robots obstáculo y los robots objetivo. Como actualmente dicho módulo no puede implementarse con hardware, se simula que se recibe el estado de todos los robots. Estas información se recibirá por medio del tipo de mensaje Map, que únicamente portará valor en las variables targets y obstacles.

- **Uav_position.** La subscripción a dicho topic le permitirá conocer al nodo cual es la posición del UAV, enviada por el nodo del flight control system. Al igual que en el topic anterior, la posición del UAV es transmitida a través de un mensaje del tipo Map, en el que únicamente tendrá valor la variable "DronePosition".

- **Map_state.** El nodo perception será el encargado de publicar en éste topic, la información recogida del mundo exterior, es decir, el estado de los diferentes robots que el propio UAV ve o siente a través de los sensores. En nuestro caso como no nos valemos del hardware para utilizar dicho módulo, se envía la información recibida del modulo de simulación. En futuras implementaciones, esta información puede ser distorsionada o reducida, con el fin de adaptarse a lo que el UAV captará en la realidad, a través de los sensores. La información será enviada mediante un mensaje del tipo "Map", a los nodos del Flight control y al nodo del mission planner.

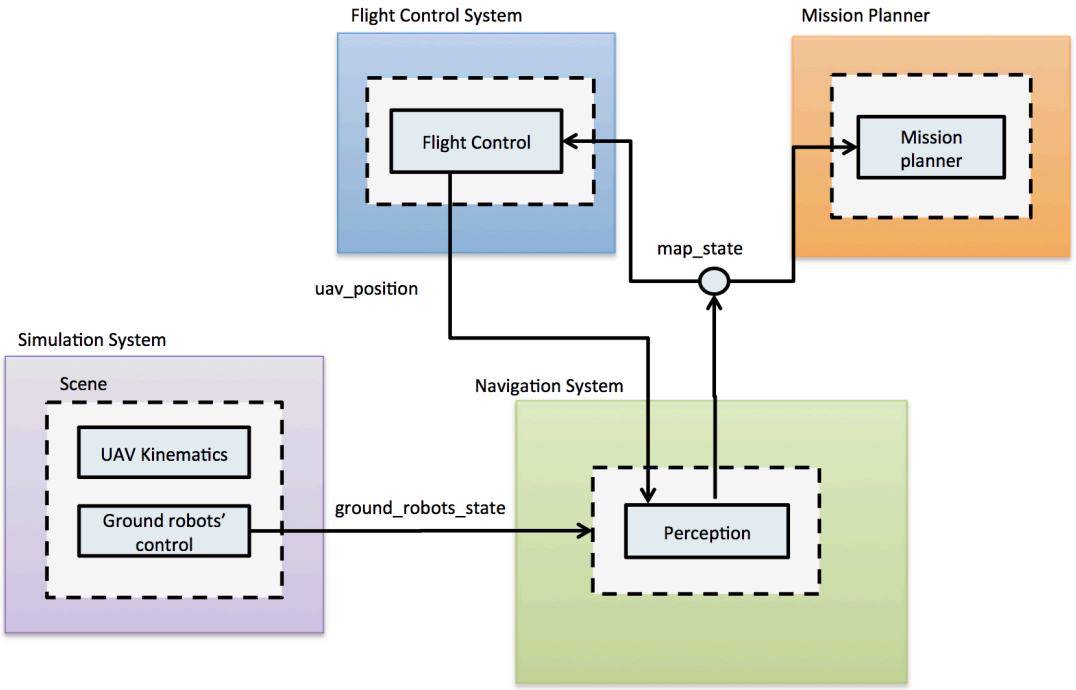

Figura 3.5. Comunicaciones del nodo "Perception".

Como se ha comentado anteriormente, para pruebas futuras sin tener el módulo del Navigation system implementado con los diferentes sensores hardware del UAV que recojan la información, a los datos obtenidos de la simulación de la escena se les podrá aplicar diversos algoritmos que funcionen como filtros y por tanto reduciendo la información que percibe el UAV y crear una simulación de una manera más cercana a la realidad.

3.6 Flight Control system

Módulo encargado de llevar a cabo las acciones demandadas por el mission planner, mediante la descomposición de dicha acción en acciones de bajo nivel, de manera que sean más entendibles por el UAV. El flight control se sirve de los datos enviados por el Navigation system para realizar dichas acciones.

El mission planner le envía la decisión de realizar una acción que puede tomar uno de los dos valores siguientes "Get Close" y "Touch Robot". Estas dos acciones realizan la siguiente función:

- **Touch Robot**: Se encargará de tocar a un determinado robot objetivo, tantas veces como sea indicado. Esta función poseerá de manera interna una máquina

de estados relativa a la posición del UAV y la posición del robot a tocar. Los diferentes estados por los que puede pasar el UAV son los siguientes:

- *Reaching*: Es una acción que se encarga de mover el UAV lo más cercano a un robot seleccionado.

- *Descending*: Con esta acción se realiza el descenso vertical del UAV para posteriormente realizar la acción de tocar al robot.

- *Magnet*: En este estado se realiza la acción de tocar al robot indicado. Esta acción consiste únicamente en posicionar a una posición lo relativamente cerca del robot objetivo para que sus sensores magnéticos se activen y se este perciba que ha sido tocado por el UAV.

- *Ascending*: Una vez se ha tocado el robot, esta acción realiza el ascenso del UAV para dejarle en una altura deseada.

- **Get Close**: Acción que se encarga de decir al UAV que se mantenga cerca de cierto robot. Para cumplimentar dicha acción el módulo se basa en la acción de bajo nivel "Reaching" que se han visto anteriormente, y la cual posiciona al UAV en una posición suficientemente cerca del robot para poder actuar sobre éste.

Para la implementación de este módulo, se utilizará un nodo ROS, que llevará el nombre de "FlightControl". Las comunicaciones con el resto de módulos se realizarán mediante el intercambio de mensajes en diferentes temas como se muestra en la Figura 3.6, estos temas se detallan a continuación:

- **Uav_position.** Este nodo será editor de dicho tema, a través del cual enviará la posición del UAV, con ello realizará la comunicación con los otros dos nodos que necesitan esta información, que son el nodo del Simulation system y el Navigation system. La posición del UAV será enviada través de un mensaje del tipo Map, en el que únicamente tendrá valor la variable "DronePosition".

- **Decision.** Mediante la subscripción al topic "Decision" se recibirá la acción, elegida por el Mission planner, que se debe ejecutar. Esta acción podrá ser una de las dos que hemos visto, "Touch" o "Get Close". Dicha información será enviada en un mensaje del tipo "Decision".

- **Control_state.** A través del siguiente topic el nodo Flight control enviará como editor el estado en el que se encuentra, al nodo mission planner, con el fin de que este le envíe o no acciones a ejecutar. Este puede indicar que está libre mediante el valor "FREE" o que esta ocupado con el valor "BUSY". Dicha información será enviada en un mensaje del tipo "FlightControlState".

- **Map_state.** Con la subscripción a este topic se realizará la comunicación con el nodo del Navigation system, pues por él se recibirá la información del mundo percibida a través de los diferentes sensores o la información simulada de la escena que nosotros deseemos. Esta información ser recibirá en un mensaje del tipo "Map".

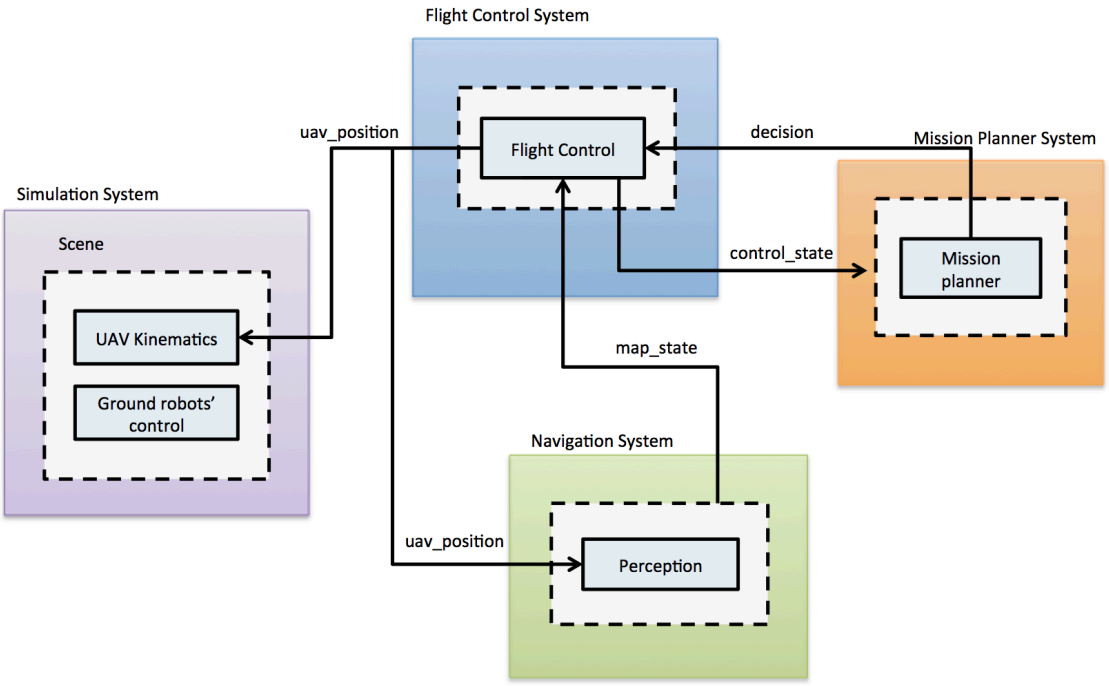

Figura 3.6. Comunicaciones del nodo "Flight Control".

3.7 Simulation System

Es el módulo que debe llevar a cabo la simulación de todos los agentes que intervienen en la arena del concurso. El fin de este módulo es ver el funcionamiento correcto de todo el conjunto de la arquitectura de control.

Gracias a la simulación del todos los agentes, se pueden ver las diferentes estrategias que se seguirán para la construcción de los otros módulos, en especial el mission planner, pues si se anula este módulo y es el propio usuario el que actúa como responsable de las acciones a tomar ,permitiendo construir el mission planner más adecuado a las necesidades del problema a resolver.
Este módulo tiene dos partes diferenciadas, la escena que realiza la simulación de movimientos de los diferentes agentes y será denominada "Scene", y la parte que recrea los objetos a visualizar, denominada "Visualization" , cuyos datos generados serán enviados a la herramienta Rviz para que esté los muestre gráficamente en un modelo tridimensional.

Al igual que todos los módulos vistos anteriormente éste será implementado como un nodo "ROS" que se llamará "Simulation". Se puede ver la representación del nodo Simulation en la Figura 3.7. A continuación se verá la implementación de este nodo, entrando en detalle en cada una de las dos partes que lo componen.

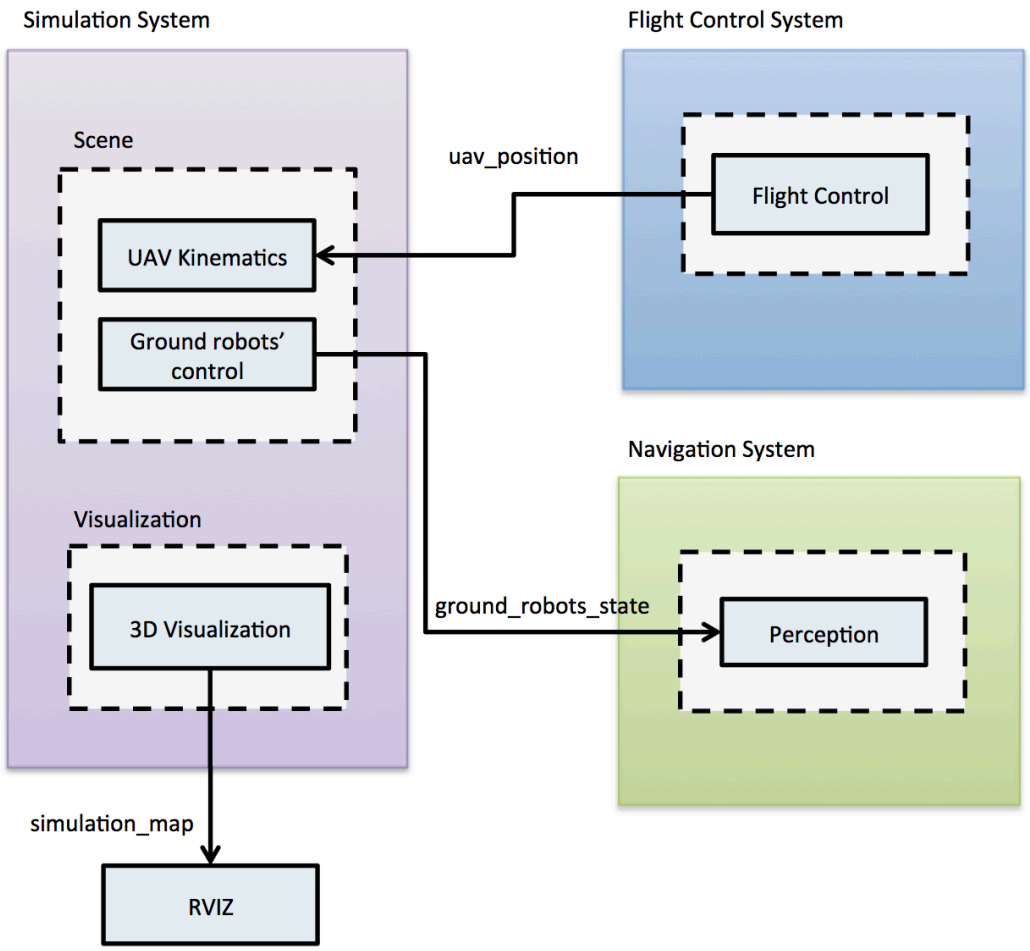

Figura 3.7. Comunicaciones del nodo "Simulation".

3.7.1 Scene

En esta parte del módulo se lleva a cabo la simulación de los diferentes agentes que intervienen en la simulación. Esta dividida en dos partes, una parte esta dedicada a la simulación del movimiento del UAV dependiendo de la información recibida desde el flight control system y la segunda parte, la cual simula los diferentes robots de tierra, basándose en el comportamiento de éstos.

- **UAV Kinematics**. La simulación del movimiento de éste es muy básica, únicamente se realiza el movimiento dependiendo de las ordenes recibidas por el flight control system. Se realiza una subscripción al topic Uav_position, por el que se recibirá una mensaje del tipo "Map", únicamente con valor en la variable "DronePosition". Este valor será una posición en coordenadas cartesianas, (x,y,z), en el que "x" e "y", serán la posición del UAV en el plano z = 0 y "z" será la altitud del UAV. Posteriormente esta información será utilizada por la parte de visualización para posicionar el marcador que represente el UAV y enviarlo al programa Rviz.

- **Ground Robots Control**. En esta parte se realiza la simulación del comportamiento de los robots objetivo y de los robots obstáculo. Se encarga de llevar a lo largo del tiempo la posición, la dirección, estado de cada uno de los robots. Éstos robots se implementarán como una máquina de estados, en el que cada estado será uno de los diferentes comportamientos que pueden adquirir éstos.

    - *Robots Objetivo*. Se guardan como una lista de un total de diez robots objetivo en el que cada uno de los robots puede tener unos de los siguientes estados:

        - TargetRun: Estado normal del robot en el que se actualiza su posición y dirección, en función del tiempo de simulación transcurrido.

        - Reverse: En este estado se produce el giro 180º en el sentido de las agujas del reloj. Se entra en este estado cada vez que transcurren 20 segundos o se produce una colisión.

        - TargetCollision: Estado que simula una colisión del robot. Una vez que se detecta la colisión, entra en dicho estado parándose

por completo el robot y posteriormente actualiza pasando al estado "Reverse" para girar 180º.

- TopTouch: Cuando el robot es tocado por el UAV, éste pasa al estado de "TopTouch", lo que hace girar al robot en el sentido de las agujas del reloj 180º. Se sabe que el robot es tocado por el UAV y se puede simular la acción del sensor magnético, por la posición relativa entre éste y el UAV, si se encuentran a una distancia, en el eje z, menor que a una distancia dada, se pasa a este estado.

- TargetNoise: Cada cinco segundos el robot entra en dicho estado, que le hace variar su dirección un ángulo aleatorio entre 0 y 20º mientras continua avanzando. Una vez a girado el ángulo determinado, vuelve al estado "TargetRun".

No se contempla el estado en el que un robot supera los limites de la arena y es eliminado, únicamente la referencia a este robot dentro de la lista es eliminada. Los estados y la causa de transición entre éstos se muestra en la Figura 3.8.

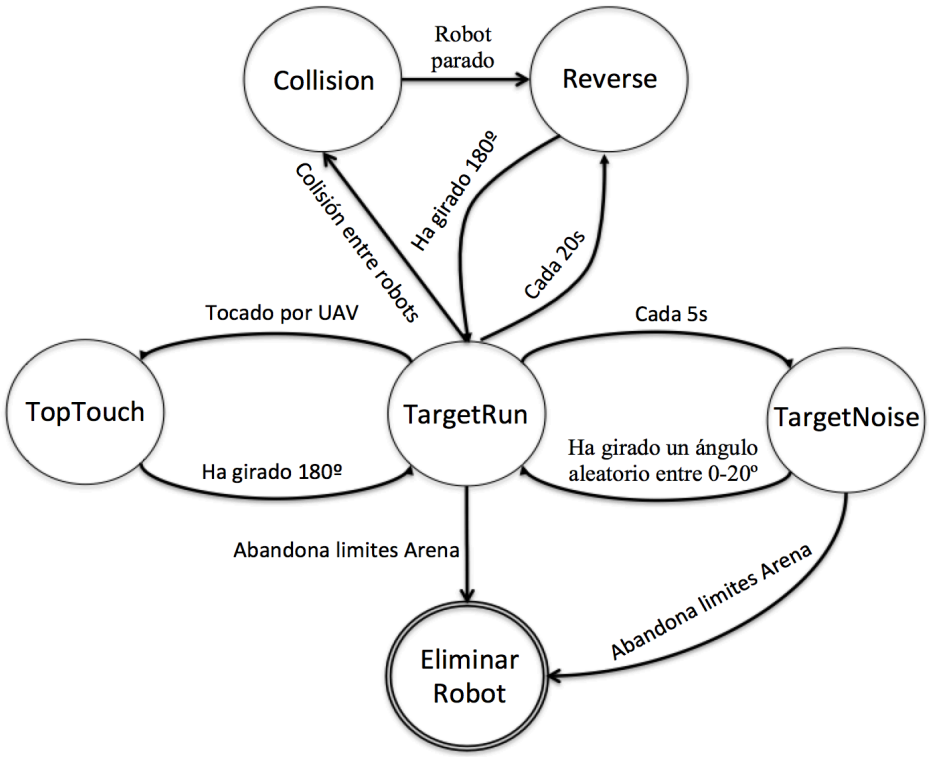

Figura 3.8. Estados y transiciones del robot objetivo.

- *Robots Obstáculo*. Al igual que los robots objetivo, están implementados se guardan en una lista y tienen únicamente dos estados.

    - ObstacleRun: Al igual que el estado de del robot objetivo, "TargetRun", es el estado normal de los robot obstáculo en el que se actualiza su posición y dirección, en función del tiempo de simulación transcurrido. Éstos girarán en círculos en el sentido de las agujas del reloj, dentro de la arena.
    - ObstacleCollision: Estado que simula una colisión del robot. Una vez que se detecta la colisión, entra en dicho estado parándose por completo el robot, espera un determinado tiempo vuelve a comprobar que no sigue colisionado y posteriormente vuelve al estado "ObstacleRun"

    A diferencia de los robots objetivo estos no son eliminados, pues siempre giraran en círculos dentro de la arena, sin salir de los limites de ésta. Los estados por los que pasa este tipo de robots se han representado en la Figura 3.9.

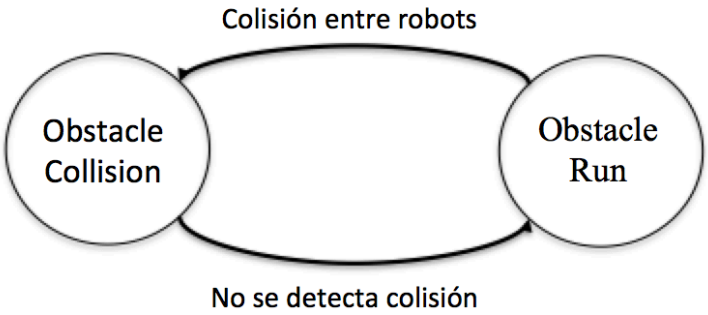

Figura 3.9. Estados y transiciones del robot obstáculo.

El estado y la posición de los diferentes robots presentes en la escena, es enviado a través del topic "ground_robots_state", mediante un mensaje del tipo "Map" en el que únicamente tendrán valor las variables "targets" y "obstacles", los cuales son listas del tipo "GroundRobotsState", en el que cada elemento contendrá información sobre los diferentes robots. La información que aquí es enviada será recogida por el nodo perception, como se ha visto anteriormente.

3.7.2 Visualización

La otra parte que conforma el módulo del Simulation system, es la parte de visualización. Se encarga de crear los objetos, denominados marcadores, que se envían a Rviz para su visualización. Inicialmente se construyen los marcadores, con las formas, tamaños, colores y posiciones deseadas, y según transcurre la simulación se encarga de actualizar la posición de cada uno de los marcadores, obteniéndola de la parte de la escena dónde se ha realizado la simulación

Primeramente es necesario entender que es un marcador y los valores que tiene éste, para posteriormente ver los diferentes marcadores construidos. Un marcador es un tipo de mensaje de ROS, el cual describe un objeto que puede ser representado visualmente por Rviz, para que este sea representado es necesario establecer ciertos valores que posee. Se detallan a continuación los elementos que se usarán para la construcción de los marcadores:

- **NS(NameSpace).** Es un identificador, que sirve para agrupar marcadores del mismo tipo. Gracias a ello se puede activar o desactivar en Rviz la visualización de los marcadores que pertenecen al mismo grupo.

- **Id.** Cada marcador de un mismo namespace, es necesario que lleve un identificador único. Esto es debido por el tipo de acción que se quiere hacer sobre cada uno de los diferentes marcadores, con el fin de actuar sobre el marcador que se desea.

- **Type.** Es el tipo de marcador, es decir, la forma geométrica de éste. Estos pueden ser de los siguientes tipos: flecha, esfera, cubo, línea, puntos, lista de líneas, lista de cubos, lista de esferas, texto orientado a la vista principal y lista de triángulos.

- **Action.** Es la acción que se debe hacer cuando se visualice el marcador. Esta puede ser añadir/modificar o eliminar el marcador.

- **Scale.** Por medio de este valor se indica el tamaño del marcador. Los valores a indicar son (x, y, z) es decir, el tamaño en cada uno de los ejes de coordenadas. Cada unidad que se le asigna a cada uno corresponde a 1 metro, en la realidad.

- **Pose.** Posición del marcador a mostrar. Se indica dando valor a los distintos elementos de los ejes de coordenadas (x, y ,z).

- **Color.** Color del marcador. Se indica dando valor a los elementos r/g/b/a con valores en el rango de [0,1].

- **Points.** Usado únicamente para los marcadores de tipo puntos, líneas y listas de cubos esferas o líneas. Sirve para indicar la posición de cada uno de los elementos que conforman este marcador.

Los diferentes marcadores que representaran los agentes y todo el conjunto de la escena son los siguientes,:

- **UAV.** Este estará formado por 7 marcadores distintos. Todos ellos con el mismo namespace, "uav" y con un identificador único, numerados desde el 0 hasta el 6. Cuatro de ellos tendrán forma cubica, simulando las hélices del UAV, dos líneas formando una cruz y representando los ejes de las hélices y un cubo central, simulando el cuerpo del propio UAV. Todos ellos de color gris translucido. En la Figura 3.10 se muestra a representación de este marcador.

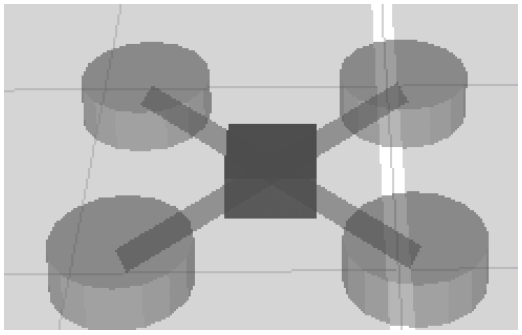

Figura 3.10. Representación en Rviz del marcador UAV.

- **Robots objetivo**. Todos ellos estarán agrupados bajo el mismo namespace "target_robots". Cada uno con un identificador único desde el 0 al 9. Poseerán forma de cubo y tendrán color blanco. En la Figura 3.11 se muestra su representación en Rviz:

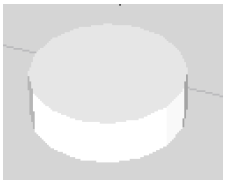

Figura 3.11. Representación en Rviz del marcador de los robot objetivo.

- **Robots obstáculo.** Formados por dos marcadores cilíndricos uno más ancho y bajo y otro más estrecho y alto simulando la altura de éstos. Estarán bajo el mismo namespace, "obstacle_robots", con un identificador único desde 0 hasta 8, en el que cada 2 dos identificadores de marcador formarán un robot. Éstos tendrán color negro. En la Figura 3.12 se muestra la representación de éstos
  .

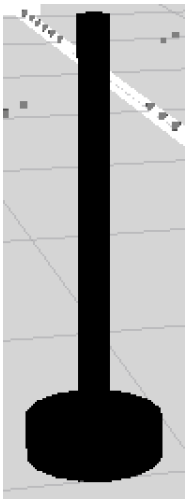

Figura 3.12. Representación en Rviz del marcador de los robot obstáculo.

- **Líneas arena.** Son 6 los marcadores que conforman todas la líneas de la arena. Estas se formarán indicando el punto de inicio y el punto final, respecto al eje de coordenadas cartesianas. Todas ellas estarán agrupadas bajo el namespace "arena_lines". A excepción de la línea ganadora, que irá de color verde y la línea opuesta a ésta, que irá de color rojo, el resto de líneas tendrán color blanco. En la Figura 26 se muestran parte de las líneas de la arena representadas.

- **Líneas de seguimiento de robots objetivo.** Formado por un marcador del tipo puntos. Se creará un marcador por cada punto indicado en el atributo "Points" mediante la tupla (x,y), que representará cada una de las posiciones por las que ha pasado el robot objetivo. Éstos puntos estarán espaciados entre sí 0.3m y la línea de seguimiento tendrá un total de 50 puntos de longitud. El color de los puntos será negro. En la Figura 26 se muestra el rastro de los diferentes robots objetivo.

Estos marcadores construidos son posteriormente enviados mediante un topic, denominado "simulation_map", del cuál estará escuchando el programa Rviz, con el fin de representar éstos gráficamente. La representación de todos de todos los marcadores se muestra en la Figura 3.13.

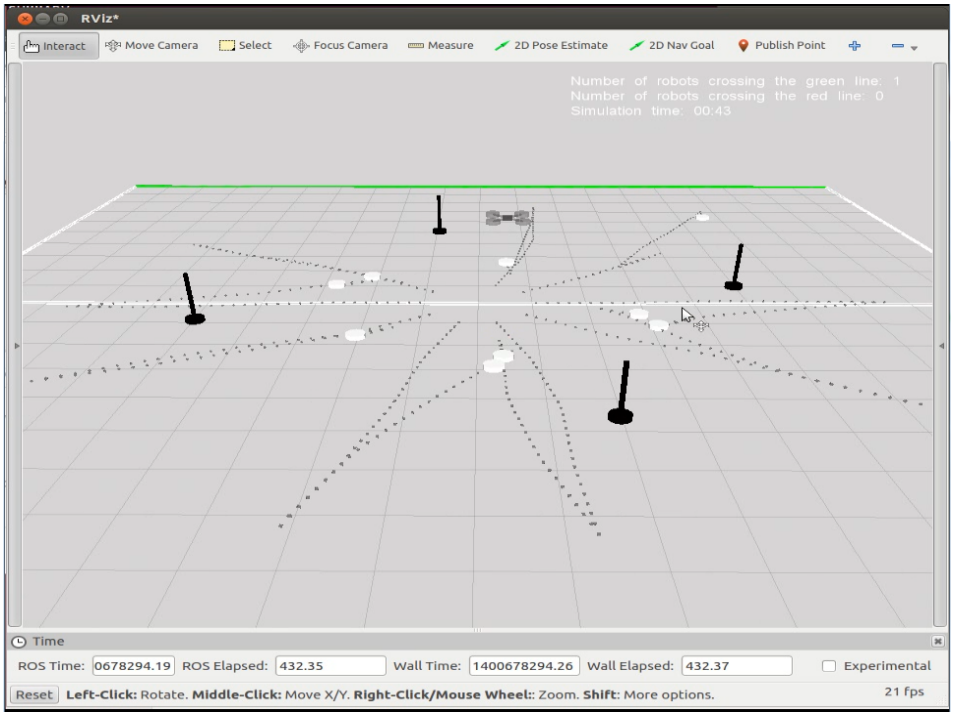

Figura 3.13. Representación en Rviz de los diferentes marcadores.

Finalmente, tras haber visto los diferentes módulos que forman la arquitectura software, en la Figura 3.14 se puede ver una representación de la construcción final de ésta y la interconexión entre los todos los módulos.

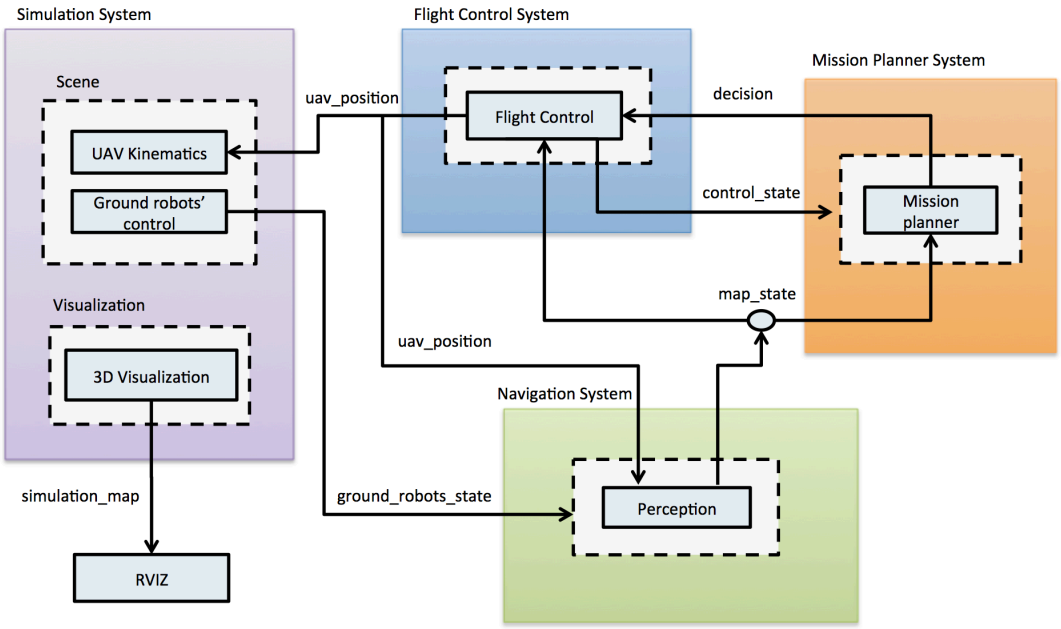

Figura 3.14. Conexión de los módulos que forman la arquitectura software.

4. VALIDACIÓN

En este capítulo se describe el proceso llevado a cabo para certificar que el simulador cumple con las objetivos propuestos. El proceso esta compuesto de diferentes verificaciones que prueban la validez del simulador.

4.1 Pruebas de validación

La validación del la arquitectura software construida, se ha llevado a cabo por medio de la simulación de todos éstos componentes, gracias a la ayuda de diferentes herramientas proporcionadas por ROS, se puede comprobar el correcto funcionamiento de todos los módulos

Primeramente era necesario comprobar la correcta construcción de todos los nodos de ROS por medio de la herramienta "rqt_graph". Esta herramienta, una vez están iniciados todos los nodos, muestra mediante un gráfico las diferentes conexiones que se generan entre todos los nodos. El gráfico generado por dicha herramienta es el mostrado en la Figura 4.1.

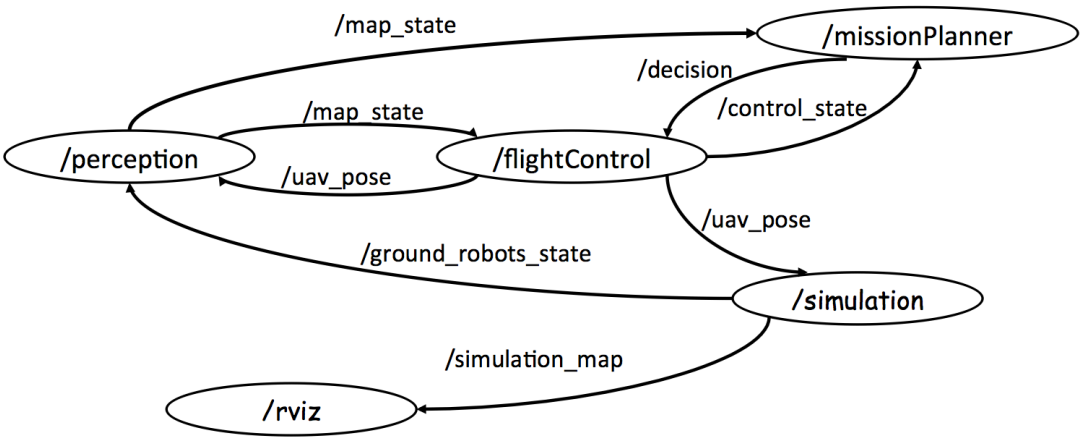

Figura 4.1. Gráfico generado con la herramienta rqt_graph.

Se comprueba que los nodos están conectados entre si mediante los diferentes topics construidos, dando por válida la construcción de la arquitectura con ROS, ya que se puede apreciar que los diferentes elementos están conectados correctamente.

Posteriormente es necesario comprobar cada uno de los topics, por medio de la visualización de los diferentes mensajes escritos en cada uno de ellos. Se comprueba que éstos mensajes son los correctos y el contenido de ellos es el que se espera. Esta comprobación se realiza mediante la herramienta "rostopic", pues permite escuchar del topic que se desee. Esta escucha se realiza por medio del comando "echo" seguido del topic correspondiente que se pretende realizar. En la Figura 4.2 se muestra un ejemplo de escucha del topic "decisión", en el que se ha escrito un mensaje del tipo "Decision" y se observan los valores del campo de este mensaje

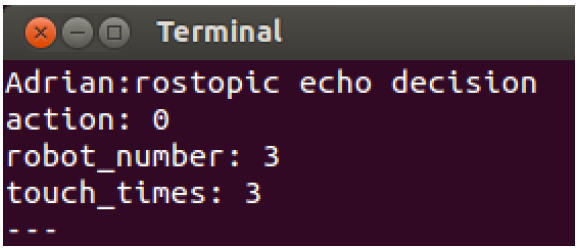

Figura 4.2. Salida generada en la escucha del topic "decisión".

El método seguido para comprobación de cada uno de los topics es el siguiente:

- Inicialmente se realiza la escucha del topic ejecutando a través de la línea de comandos, el programa de la siguiente manera "*rostopic echo nombre_del_topic*", una vez iniciado, se encuentra a la espera hasta que se escriba algún mensaje para su visualización.

- Seguidamente se realiza la ejecución de los nodos que escriben y escuchan en el topic correspondiente, para ver que éstos mensajes son de verdad escritos y escuchados por los nodos.

Las pruebas realizadas siguiendo el método anterior son las siguientes:

- **Comprobación de las conexiones entre los nodos Mission planner y Flight control.**
  Se realiza la escucha de los topic "decisión" y "control state" y posteriormente se ejecutan los nodos flight control system y mission planner. Una vez éstos están iniciados, se espera que se escriba primeramente en el topic de "control_state", un mensaje del tipo "FlightControlState" con el valor "FREE" indicando al mission planner que éste modulo está libre y espera una acción. Una vez recibido éste por el mission planner, se encargar de seleccionar una acción y escribir un mensaje de tipo "Decision" en el topic "decisión". Cuando

el mensaje es recibido por el flight control system, éste escribe un mensaje al mission planner indicándole que esta ocupado, por medio del valor "BUSY" en un mensaje del tipo "FlightControlState". Se comprueban que todos los mensajes son enviados por medio de la herramienta "rostopic", posteriormente recibidos, pues el nodo que lo recibe ejecuta una acción, y todo ello en el orden descrito.

- **Comprobación de las conexiones entre los nodos Flight Control, Simulation y Perception.**
  La finalidad de esta prueba es comprobar que el nodo perception recibe los mensajes de los nodos flight control y simulation y posteriormente genera el mensaje "Map". Inicialmente se realiza la escucha de todos los topics que intervienen en la prueba, éstos son: "uav_position" , "ground_robots_state" y "map_state". A continuación se ponen en funcionamiento todos los nodos, pues para que el nodo del flight control funcione, necesita que el mission planner le diga que realice cierta acción.

  Se comprueba que se escriben los mensajes en los topics "uav_position" y "ground_robots_state", mediante la escucha de éstos con "rostopic echo" y son recibidos por el nodo perception. La recepción de los mensajes por el módulo perception se realiza mediante la comprobación de la escritura de un mensaje del tipo "Map" en el topic "Map_state", pues hasta que no recibe los mensajes de "uav_position" y "Ground_robots_state" no puede generar dicho mensaje.

- **Depuración de código de los diferentes nodos ROS.**
  Para depurar el código era necesario ver cual era el contenido en cada uno de los mensajes que escribían los diferentes nodos. Por ello, cuando algo no funcionaba como se esperaba, inicialmente se realizaba la escucha del topic correspondiente, por el cual debía escribir el nodo a depurar, y se ponía a ejecutar dicho nodo, comprobando que los valores del mensaje que escribía eran los correctos.

Para la comprobación final de la correcta conexión entre todos los módulos que conforman la arquitectura software, se usó la visualización con Rviz. Si todos los datos eran recibidos y representados de manera correcta por Rviz, se daba por valido las conexiones entre los diferentes módulos.

Una vez se tuvo validada las conexiones entre todos los nodos, se realizo la validación del correcto comportamiento del software. Esta validación se llevo a cabo visualmente,

mediante la representación en Rviz. A continuación se describen los diferentes detalles que se tuvieron en cuenta para dar por buena construcción de la arquitectura:

- **Representación correcta de todos los agentes y la arena.** Mediante la visualización se comprueba que todos los agentes que intervienen en la simulación, están correctamente representados, es decir, que su forma geométrica se adapta a la realidad así como su tamaño, y que se muestren tal y como se han implementado. Así como las diferentes líneas que forman la arena.

- **Comprobación de un comportamiento correcto de los robots.** Mediante la visualización se comprueba el correcto funcionamiento de los robots obstáculo. Primero, desde el inicio de la simulación, se comprueba que estos se muevan en círculos dentro de la arena, en el sentido de las agujas del reloj. Posteriormente, cuando sufran alguna colisión, no se solapen gráficamente con el otro robot colisionado y además tengan el comportamiento adecuado, es decir, que se paren y mantengan su posición hasta que el robot con el que han colisionado se mueva y no exista colisión, en cuyo caso continuarán con la marcha.

  Al igual que los robots obstáculo, se comprueba visualmente el comportamiento de los robots objetivo. Para ver una buena percepción de los cambios de comportamiento de los robots, se les hace que cambie de color cada vez que sufren un cambio de estado. Se comprueba que los diferentes comportamientos de éste sean los siguientes:

  - Cuando se inicia la simulación los robots deben moverse recto, en todas las direcciones de la arena, manteniendo el color blanco.

  - En caso de sufrir una colisión, el robot no puede solaparse gráficamente con el otro robot. Además, se comprueba que cambian su color a naranja indicando que han colisionado y deben girar en sentido de las agujas del reloj 180º. Una vez han girado se comprueba que vuelven a cambiar el color a color blanco.

  - Cada 20 segundos los robots debed cambiar su dirección 180º en dirección a las agujas del reloj, cambiando su color de blanco a naranja. Al igual que en el comportamiento anterior, una vez han girado se comprueba que vuelven a cambiar el color a color blanco.

  - Cada 5 segundos los robots variaran su dirección un valor aleatorio entre 0 y 20º. Este cambio de comportamiento se ha representado gráficamente

mediante el cambio de color del robot, a color amarillo. Una vez se produce el cambio de dirección, vuelven a ser de color blanco.

- Cuando el robot es tocado por el UAV, para tener conocimiento de si éste ha recibido la orden, se le hace cambiar de color a color naranja. Al igual que en los anteriores, hasta que no gira 180º en el sentido de las agujas del reloj, no debe volver a color blanco.

- **Comprobación del rastro de los robot.** Se comprueba visualmente que según avanzan los robots objetivo se representa gráficamente su rastro, mediante una serie de puntos que deben estar separados entre sí. Además, se comprueba que la longitud de ésta no sea mayor a 50 puntos, y que los puntos más antiguos del rastro se van eliminando una vez se excede esta cantidad.

- **Comprobación eliminación de robots**. Cuando uno de los robots excede los limites de la arena, se comprueba que se produce la eliminación de éste junto con el rastro que se había dibujado de su trayectoria.

- **Comprobación del comportamiento del UAV dependiendo del mission planner.** Inicialmente para ver que la representación del UAV y que la actualización de su posición se hacía correctamente, se utilizó un mission planner básico, en el que el UAV debía alcanzar la posición de los robots objetivo de uno en uno en orden cíclico, comenzando desde el robot identificado con el número 0, hasta el robot identificado con el 9 y así volviendo a comenzar.

  Posteriormente se uso la versión actual de mission planner, la cuál es mucho más compleja, ya que actualizaba también la posición de éste en el plano vertical. La finalidad era comprobar de que el UAV respondía gráficamente a las ordenes de ascenso y descenso cuando tenía por objetivo tocar a un determinado robot de tierra. Además de que cuando tocase al robot, éste cambiase de estado.

  Una vez realizadas ambas pruebas, se dio por buena la representación gráfica del UAV, en el simulador.

Tras realizar todas las pruebas anteriormente descritas, se da por buena y validada la construcción final de la arquitectura del simulador. Esta versión final es apta, de acuerdo a los requisitos del concurso IARC para probar diversos mission planner que permitan al UAV alcanzar el objetivo de la misión.

5. CONCLUSIONES

El objetivo de éste trabajo ha sido el diseño e implementación de un sistema software que permitiese dar soporte al desarrollo un planificador de misiones para un UAV y que sirviese de ayuda para que la elección de la estrategia a seguir por este planificador sea satisfactoria de acuerdo con los objetivos de la competición IARC. Para ello se han realizado las siguientes tareas principales:

- En la fase de análisis primeramente se hizo un estudio del concurso IARC, pues era la problemática principal y de la que surgían la mayoría de los requisitos para implementar la arquitectura de control. Para enfrentarnos a este problema se realizó un estudio de los diferentes paradigmas de programación en cuanto a inteligencia robótica, los diferentes marcos existentes de clasificación según la autonomía de los robots y los actuales avances y grupos de investigación existentes en el campo de los UAV. Todo ello permitió obtener una idea mayor del problema al que había que hacer frente y las posibles herramientas o métodos que se tenían para solucionarlo.

- El estudio realizado durante la primera fase de análisis, de las diversas materias en cuanto a robótica inteligente, sirvió de ayuda para afrontar la realización del diseño de la arquitectura de control. Para el diseño de esta arquitectura se buscó principalmente que fuera modular, con el fin de que el cambio de uno de éstos módulos no afectase al resto de los componentes y permitiera principalmente la prueba de diferentes versiones de mission planner. Finalmente se optó por un diseño basado en el modelo (Guide Navigation Control – GNC) el cual permitía esta modularidad.

- Para implementar esta arquitectura de control se eligió el sistema ROS, un adecuado sistema software para desarrollo de aplicaciones de robótica. Éste proporciona unas herramientas muy útiles para la construcción de la arquitectura de una manera modular. Además de esto, nos permitía visualizar la simulación de la arquitectura por medio de su herramienta Rviz. Esta visualización, además de servir para ver el comportamiento del mission planner, sirvió para la validación del propio simulador.

- En la fase de validación se han realizado las pruebas pertinentes, con el fin de ver que el simulador cumplía con la modularidad deseada y con todos los requisitos del concurso IARC. Finalmente tras las pruebas realizadas se consigue obtener una estructura de una arquitectura de control la cual puede ser simulada y visualizada mediante la herramienta Rviz.

Esta arquitectura ha permitido realizar la simulación de diferentes prototipos del planificador de misiones, en fases previas antes de la terminación del equipamiento completo del vehículo UAV. Con ello, ha sido posible definir estrategias de planificación cuya eficacia respecto a las reglas de la competición IARC se ha podido verificar con ayuda del entorno de simulación.

ANEXO A: Visualización del comportamiento

En este anexo se muestran ejemplos representativos de visualización de diferentes posiciones de los robots durante el desarrollo de una misión.

En la Figura A.1 se muestra la posición de los robots al inicio de la misión.

En la Figura A.2 se representa la acción del UAV cuando éste toca a uno de los robots objetivo.

En la Figura A.3 se representan los diferentes comportamientos de los robots de tierra, en amarillo aquellos que deben variar su dirección entre 0 y 20º cada 5 segundos y en naranja los que han llegado al fin del ciclo de los 20 segundos y deben girar 180º.

En la Figura A.4, se muestra la colisión entre los diferentes robots, y como el robot objetivo debe girar 180º, comportamiento representado por el cambio de color a naranja.

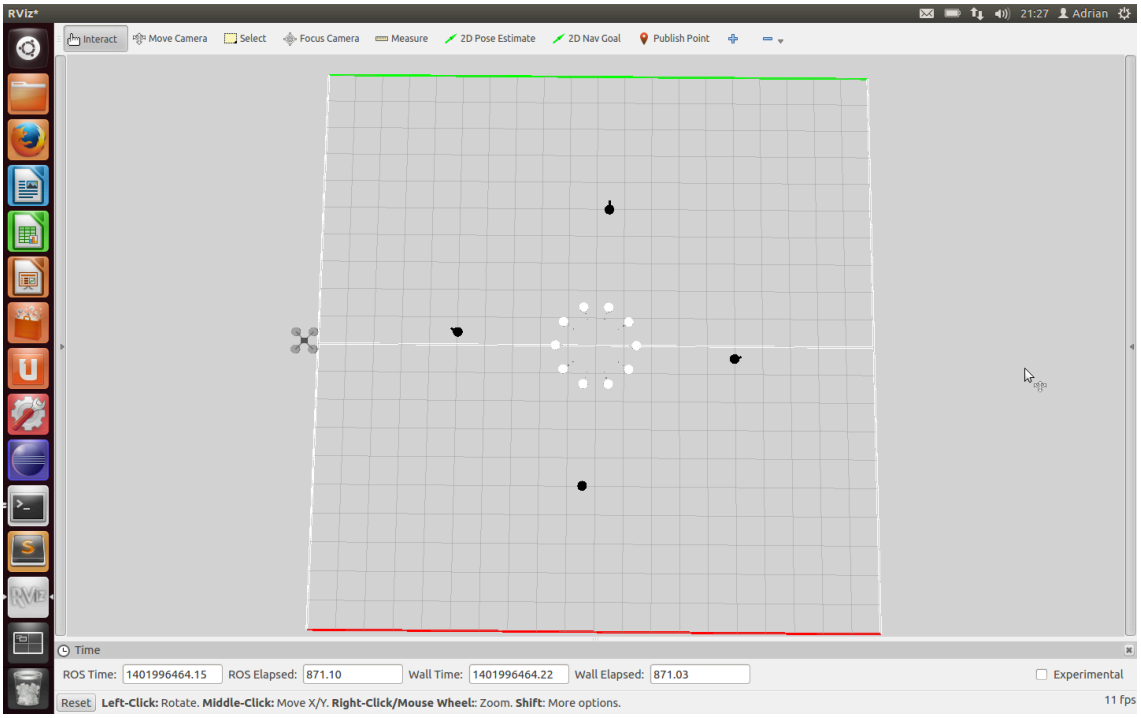

Figura A.1. Inicio de la misión IARC.

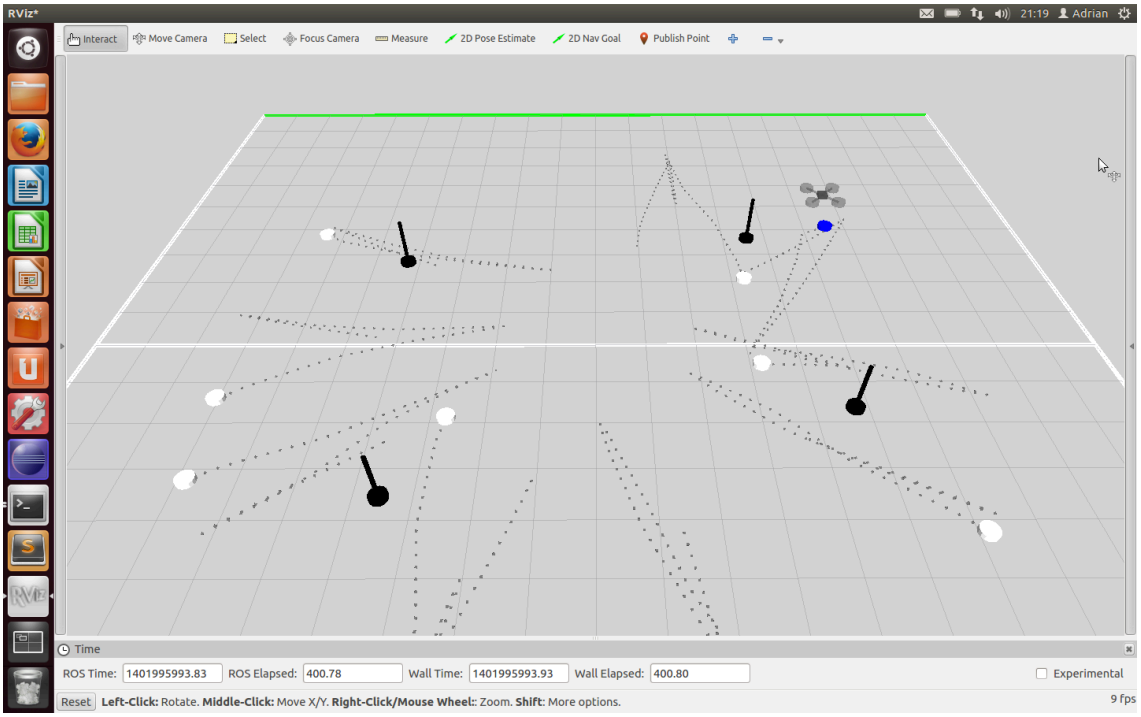

Figura A.2. Acción de tocar realizada por UAV sobre robot objetivo.

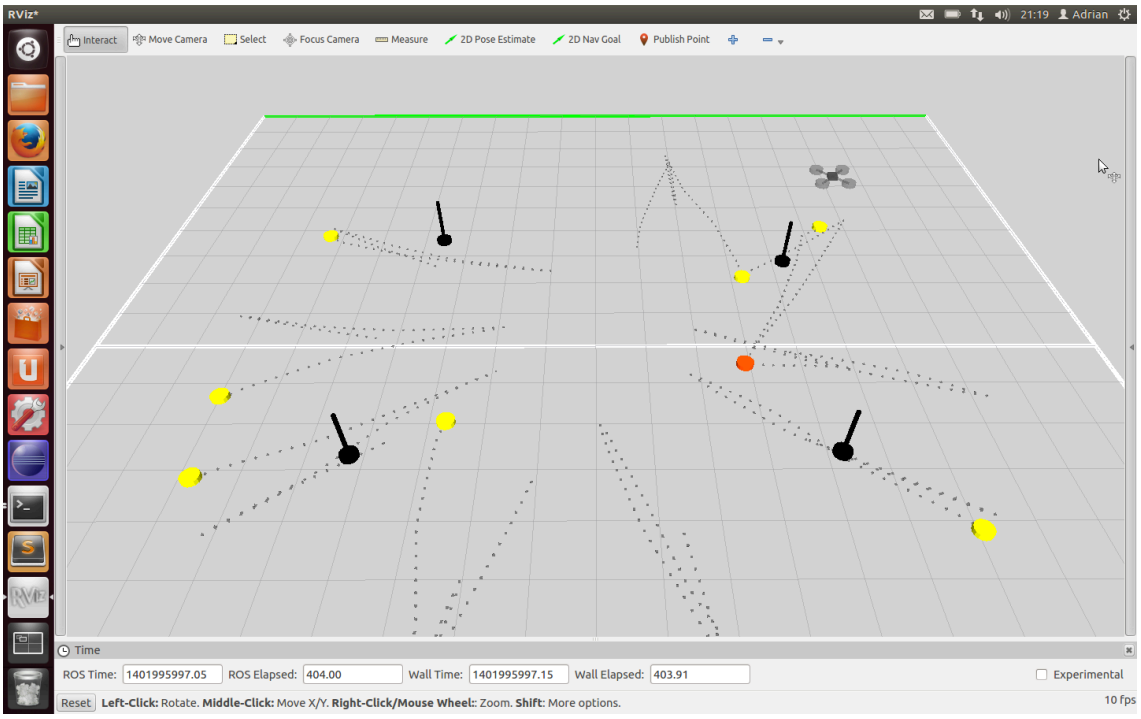

Figura A.3. Diferentes comportamientos de los robots objetivo.

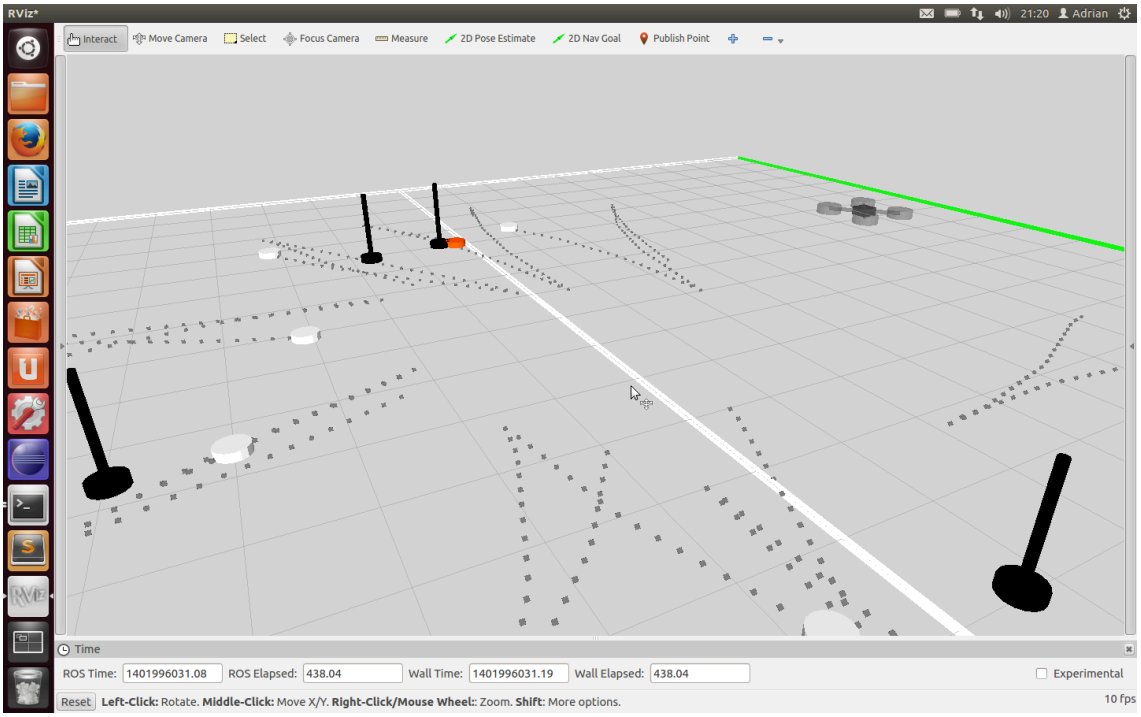

Figura A.4. Colisión entre robots.

REFERENCIAS BIBLIOGRÁFICAS

Dickerson, L. (2007): "UAV on the rise". *Aviation Week & Space Technology, Aerospace Source Book 2007*, 166(3).

Endsley, M. R. (1999): "Situation awareness in aviation systems". . *Handbook of Aviation Human Factors, Publication of Lawrence Erlbaum Associates*.

IARC (2014): "*Past IARC missions"*. [Online]. Available: www.aerialroboticscompetition.org/pastmissions.php

Kendoul (2012): "*A Survey of Advances in Guidance, Navigation and Control of Unmanned Rotorcraft Systems"*.

Murphy (2000): "*An Introduction to AI Robotics (Intelligent robotics and Autonomous Agents)"*. MIT Press.

Parasuraman, R et al. (2000): Parasuraman, R., Sheridan, T. B., and Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics*, 30(3):286–297.

ROS (2007). "*ROS – Robot Operating System"*. [Online] Available: http://www.ros.org/

Sheridan, T. B. (1992): "*Telerobotics, Automation, and Human Supervisory Control"*. MIT Press.

Visiongain (2009). "*The unmanned aerial vehicles (UAV) market 2009-2019*".

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Fri Jun 06 23:59:20 CEST 2014
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)